spring

HIBERNATE

SUNBEAM

Trainer: Nilesh Ghule

Wake up from Hibernate, Spring up!!!

Agenda

- Transaction ✓
- Hibernate entity life cycle ✓
- Dialect ✓
- openSession() vs getCurrentSession()
- Criteria and DetachedCriteria ✓
- HQL ✓
- Named Queries ✓

CRUD operations

- Hibernate Session methods
  - get() or find(): Find the database record by primary key and return it. If record is not found, returns null.
  - load(): Returns proxy for entity object (storing only primary key). When fields are accessed on proxy, SELECT query is fired on database and record data is fetched. If record not found, exception is thrown.
  - save(): Assign primary key to the entity and execute INSERT statement to insert it into database. Return primary key of new record.
  - persist(): Add entity object into hibernate session. Execute INSERT statement to insert it into database (for all insertable columns) while committing the transaction.
  - update(): Add entity object into hibernate session. Execute UPDATE statement to update it into database while committing the transaction. All (updateable) fields are updated into database (for primary key of entity).
  - saveOrUpdate() or merge(): Execute SELECT query to check if record is present in database. If found, execute UPDATE query to update record; otherwise execute INSERT query to insert new record.
  - delete() or remove(): Delete entity from database (for primary key of entity) while committing the transaction.
  - evict() or detach(): Removes entity from hibernate session. Any changes done into the session after this, will not be automatically updated into the database.
  - clear(): Remove all entity objects from hibernate session.
  - refresh(): Execute SELECT query to re-fetch latest record data from the database.

Hibernate Transaction → RDBMS

hibernate.cfg.xml:
hibernate.connection.autocommit=false

JDBC → con.setAutoCommit(false);
=
con.commit();
con.rollback();

hibernate:
↳ jdbc tx
→ jta tx
→ ...

- In hibernate, autocommit is false by default.
- DML operations should be performed using transaction.
  - session.beginTransaction(): to start new tx.
  - tx.commit() & tx.rollback(): to commit/rollback tx.
- session.flush()
  - Forcibly synchronize in-memory state of hibernate session with database.
  - Each tx.commit() automatically flush the state of session.
  - Manually calling flush() will result in executing appropriate SQL queries into database.
  - Note that flush() will not commit the data into the RDBMS tables.
  - The flush mode can be set using session.setHibernateFlushMode(mode).
    - ALWAYS, AUTO, COMMIT, MANUAL → readonly session

poor performance ← ALWAYS; default ← AUTO; only during commit ← COMMIT; setFlushMode()

- If hibernate.connection.autocommit is set to true, we can use flush to force executing DML queries.

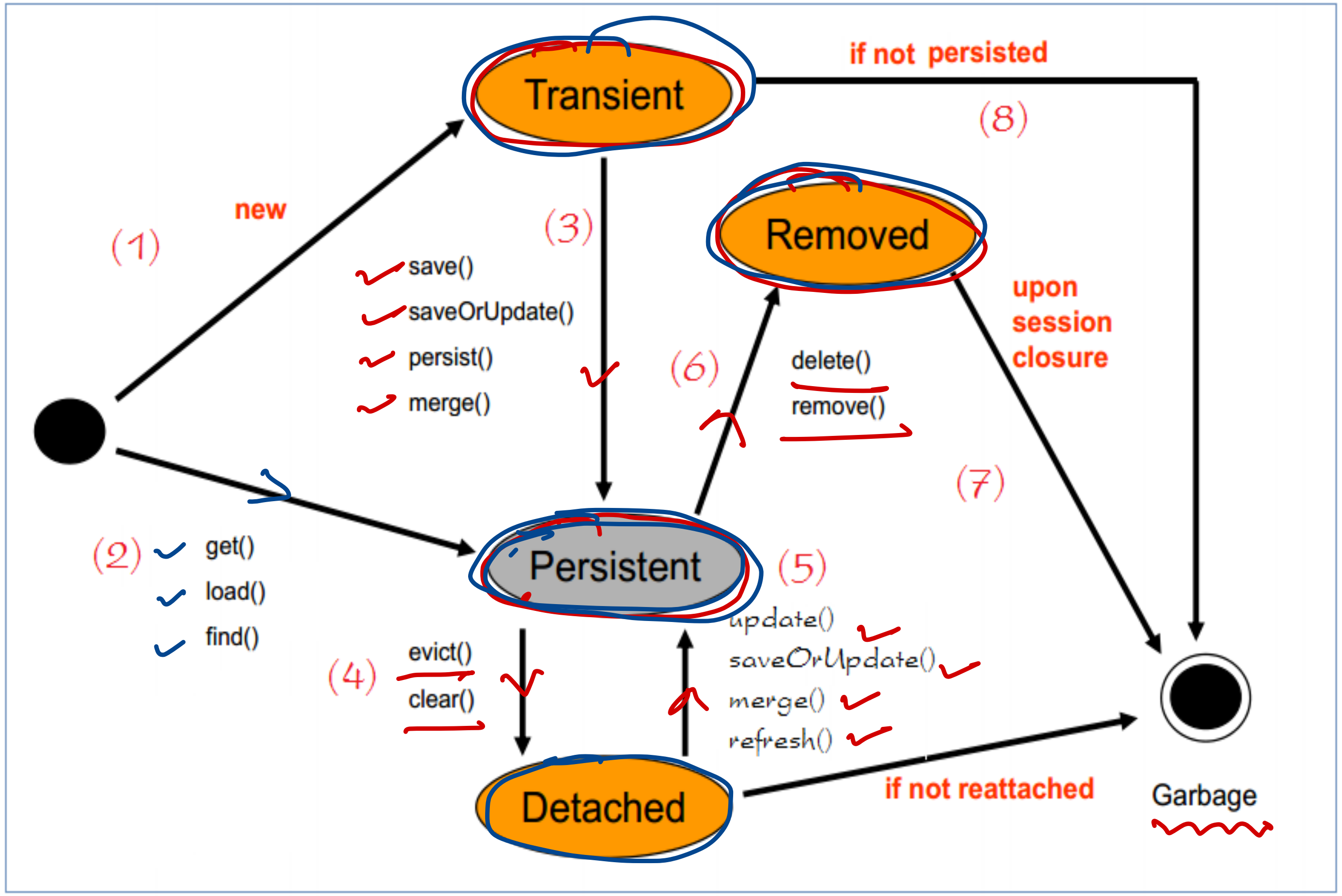
tx

Hibernate – Entity life cycle

- (1) Transient
  - New Java object of entity class.
  - This object is not yet associated with hibernate.
- (2) Persistent
  - Object in session cache.
  - For all objects created by hibernate or associated with hibernate.
  - State is tracked by hibernate and updated in database during commit.

dirty flag

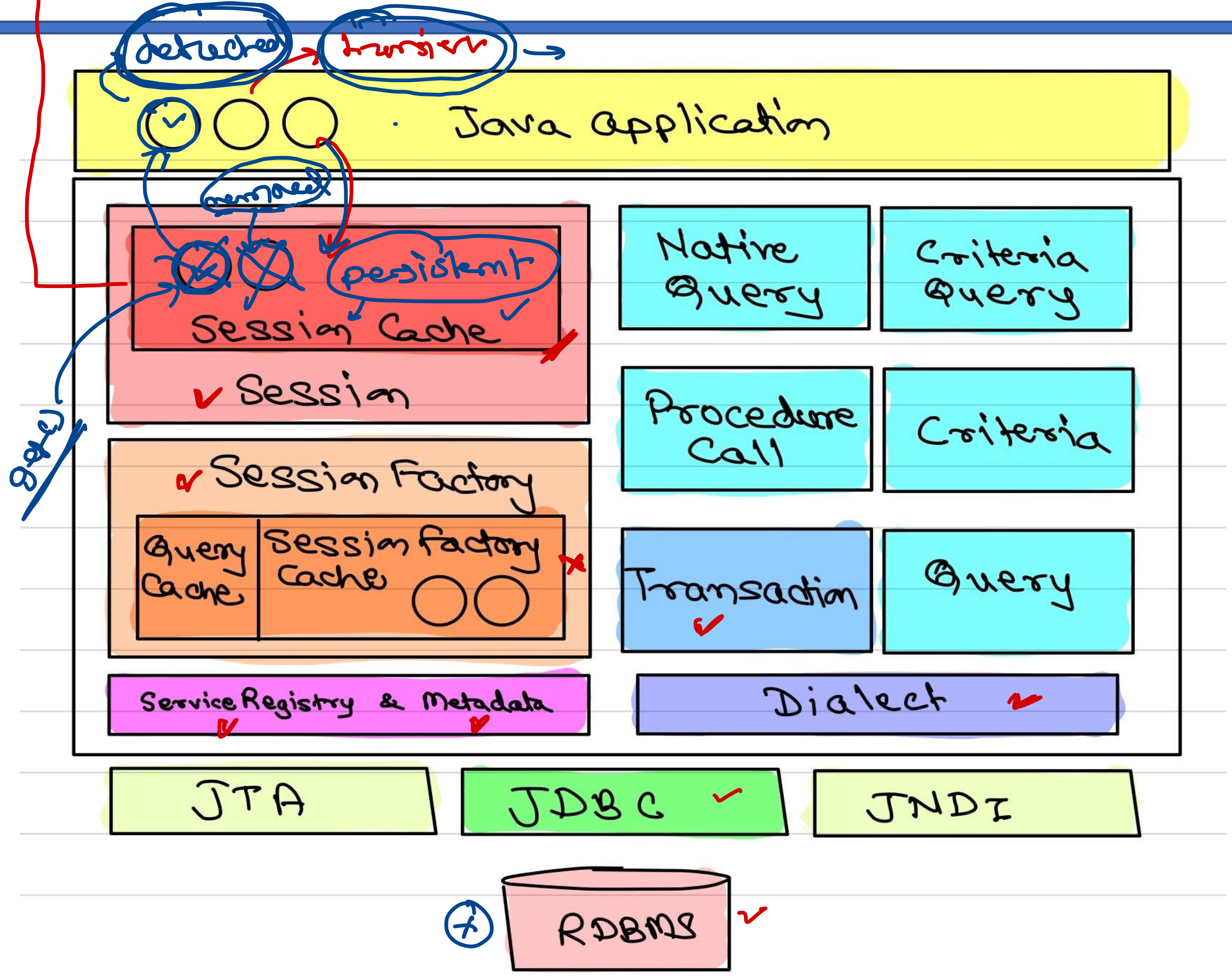
- Never garbage collected.
  - (3) Detached
  - Object removed from session cache.
- Removed
  - Object whose corresponding row is deleted from database.



Hibernate

part of session cannot be disabled.

get(), persist(), update(), remove()



- Dialect
  - RDBMS have specific features like data types, stored procedures, primary key generation, etc.
  - Hibernate support all RDBMS.
  - Most of code base of Hibernate is common.
  - Database level changes are to be handled specifically and appropriate queries should be generated. This is handled by Dialect.

    org.hibernate.dialect.
  - Hibernate have dialects for all RDBMS. Programmer should configure appropriate dialect to utilize full features of RDBMS.

hibernate.cfg.xml

openSession() vs getCurrentSession()

- openSession()
  - Create new hibernate session.
  - It is associated with JDBC connection (autocommit=false).
  - Can be used for DQL (get records), but cannot be used for DML without transaction.
  - Should be closed after its use.

S1 →

S2 →

s1 = factory.getCurrentSession();

s2 = factory.getCurrentSession();

- getCurrentSession()
  - Returns session associated with current context.
  - Session will be associated with one of the context (hibernate.current_session_context_class).
    - thread: Session is stored in TLS.
    - jta: Session is stored in transaction-context given by JTA providers (like app servers).
    - custom: User implemented context.
  - This session is not attached with any JDBC connection.
  - JDBC connection is associated with it, when a transaction is created. The connection is given up, when transaction is completed.
  - The session is automatically closed, when scope is finished. It should not be closed manually.

Criteria vs DetachedCriteria → hibernate. SELECT only

- Criteria
  - represents WHERE clause for SELECT query.
  - cr.add() -- condition and cr.addOrder() -- sort order.
  - for condition -- helper class Restrictions
  - eq(), ne(), gt(), lt(), ...
  - for order -- helper class Order → a
  - asc(), desc()
  - Created using session.createCriteria().
  - Have methods to fire SELECT query
  - cr.list(), cr.uniqueResult().

deprecate

- DetachedCriteria
  - Similar to Criteria object, but NOT associated with Session.
  - dcr = DetachedCriteria.forClass(MyEntity.class);
  - dcr.add() -- condition and dcr.addOrder() -- sort order.
  - To execute, call dcr.getExecutableCriteria(session), that gives Criteria object.
  - The Criteria object can give result using list() or uniqueResult().

Native Queries and HQL

- Ad-hoc SQL queries (on tables) can be executed in hibernate directly in hibernate.
  - NativeQuery q = session.createSQLQuery(sql);
- Hibernate recommends using HQL for ad-hoc queries.
- These queries are on hibernate entities (not on tables).
  - Query q = session.createQuery(hql);
- HQL supports SELECT, DELETE, UPDATE operation.
- INSERT is limited to INSERT INTO ... SELECT ...;

SQL → on rdbms table

HQL → on hibernate entities.

Hibernate – HQL

- SELECT
  - from Book b
  - from Book b where b.subject = :p_subject
  - from Book b order by b.price desc
  - select distinct b.subject from Book b
  - select b.subject, sum(b.price) from Book b group by b.subject
  - select new Book(b.id, b.name, b.price) from Book b
- DELETE
  - delete from Book b where b.subject = :p_subject
- UPDATE
  - update Book b set b.price = b.price + 50 WHERE b.subject = :p_subject
- INSERT
  - insert into Book(id, name, price) select id, name, price from old_books

Named queries

- Using multiple HQL queries in project will scatter them into multiple files and thus application maintenance become complicated.
- All queries related to an entity can be associated with the class using @NamedNativeQuery or @NamedQuery.
- NamedNativeQuery represent SQL query.
  - Use session.getNamedNativeQuery() to access NativeQuery.
  - Invoke methods on NativeQuery to perform appropriate operations.
- NamedQuery represent HQL query.
  - Use session.getNamedQuery() to access NativeQuery.
  - Invoke methods on NativeQuery to perform appropriate operations.
- To associate multiple queries use @NamedNativeQueries or @NamedQueries.

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>