CitiusTech

Hibernate Basics

Session 1

Version 1.2

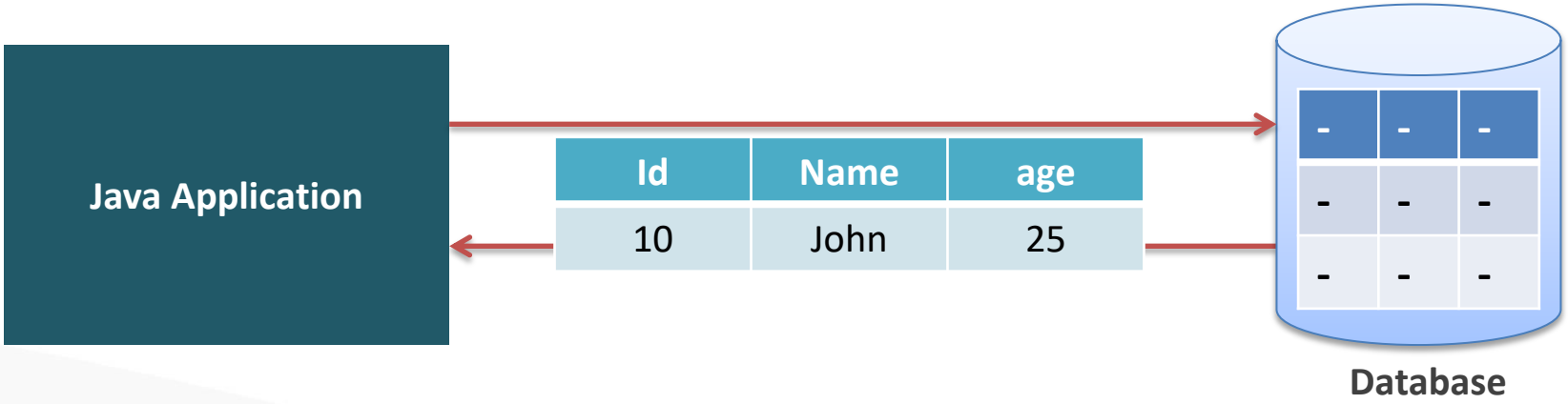
September 2018

Agenda

- **Introduction to ORM**
- What is Hibernate & Why Hibernate?
- Hibernate Architecture
- Hibernate Framework Objects
- Instance Life Cycle

Persistence in Applications

- What is Persistence?
  - Almost all applications require persistent data
  - Persistence means individual objects can outlive the application process
  - Objects state can be saved to a data store and be re-created at a later point in time
- Persistence in Java
  - Mapping and storing object instances in a relational database using Structured Query Language (SQL)



In Java, at the low level we use JDBC API for connecting to databases and perform CRUD operations

Object-Relational Impedance Mismatch

- Object-Relational Impedance mismatch means:
  - Object models and relational models do not work very well together
  - Loading and storing graph of objects using a tabular relational database exposes mismatch problems
  - Mismatch problems are as follows:
    - Granularity
    - Subtypes (Inheritance)
    - Identity
    - Associations
    - Data Navigation

A mechanism or solution is needed to overcome this incompatibility gap for proper collaboration

Object-Relational Mapping

- Object-relational Mapping (ORM and O/R mapping tool) refers to the technique of mapping data between an object model representation to a relational data model representation
- Some of the advantages of ORM tools are as follows:
  - Facilitates Domain Model Pattern - This pattern means that you model entities based on real business concepts rather than based on your database structure
  - Huge Reduction in Code - ORM tools provide a host of services thereby allowing developers to focus on the business logic of the application rather than repetitive CRUD (Create Read Update Delete) logic
  - Object Structure - Changes to the object model are made in one place. One you update your object definitions, the ORM will automatically use the updated structure for retrievals and updates
  - Rich Query Capability - ORM tools provide an object oriented query language
  - Navigation - You can navigate object relationships transparently. Related objects are automatically loaded as needed

O/R Mapping Tools

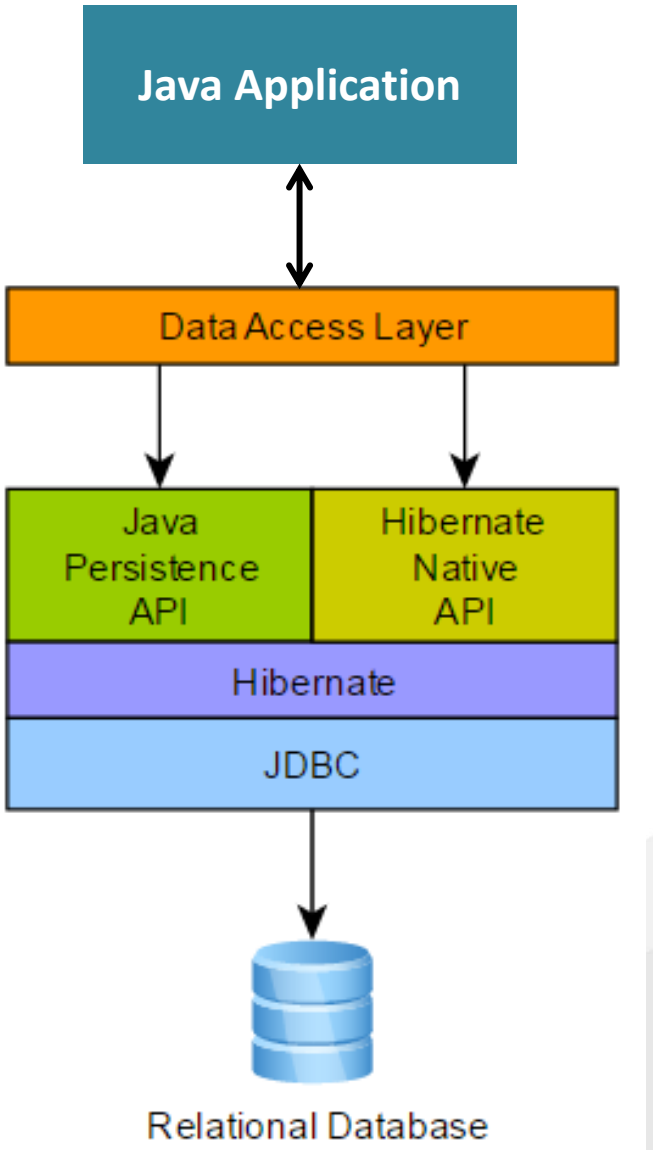
- Java ORM frameworks:
  - Enterprise JavaBeans Entity Beans
  - Java Persistence API
  - Hibernate
  - Java Data Objects
  - IBatis
  - Castor
  - TopLink
  - Spring DAO

Agenda

- Introduction to ORM
- **What is Hibernate & Why Hibernate?**
- Hibernate Architecture
- Hibernate Framework Objects
- Instance Life Cycle

What is Hibernate?

- Hibernate is an Object/Relational Mapping (ORM) solution for Java environments
- Hibernate, as an ORM solution, effectively "sits between" the Java application data access layer and the Relational Database
- It helps you to build persistent objects following common OO programing concepts:
  - Association
  - Inheritance
  - Polymorphism
  - Composition
  - Collection API for “many” relationship



Source : https://blogs.perficient.com/2017/09/19/basic-usage-about-hibernate/

Why Hibernate?

- Handles Object-Relational impedance mismatch
- Allows developers to focus on domain object modelling and not on the persistence plumbing
- Reduces development time
- Improves performance
  - High performance object caching
  - Configurable materialization strategies
- Provides sophisticated query facility
  - Criteria API
  - Query By Example (QBE)
  - Hibernate Query Language (HQL)
  - Native SQL

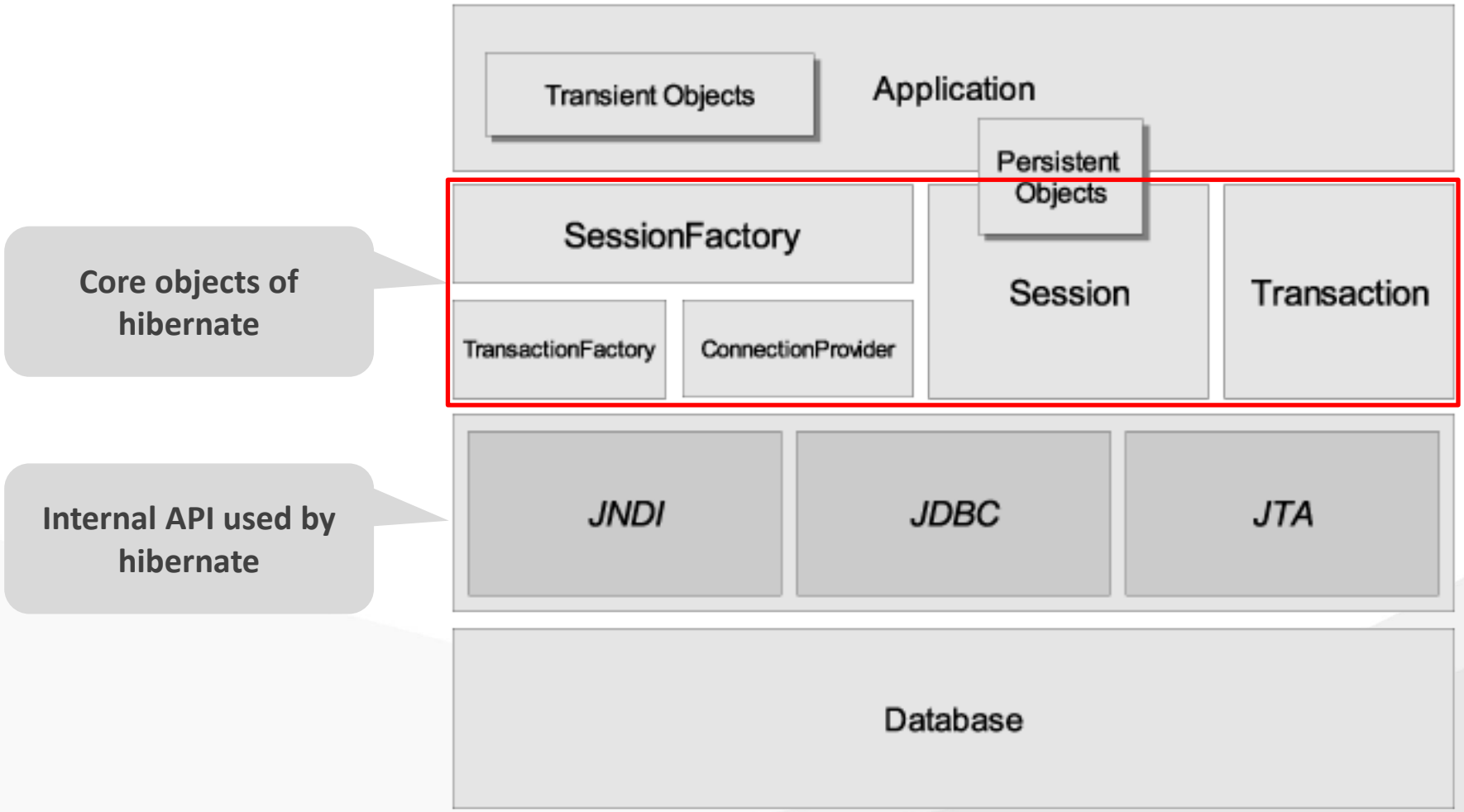
The Java application makes use of the Hibernate APIs on its domain data to load, store, query, and so on

Agenda

- Introduction to ORM
- What is Hibernate & Why Hibernate?
- **Hibernate Architecture**
- Hibernate Framework Objects
- Instance Life Cycle

Hibernate Architecture

- Following figure shows architecture that abstracts the application from the underlying JDBC/JTA APIs and lets Hibernate take care of the details



Components of Hibernate Framework

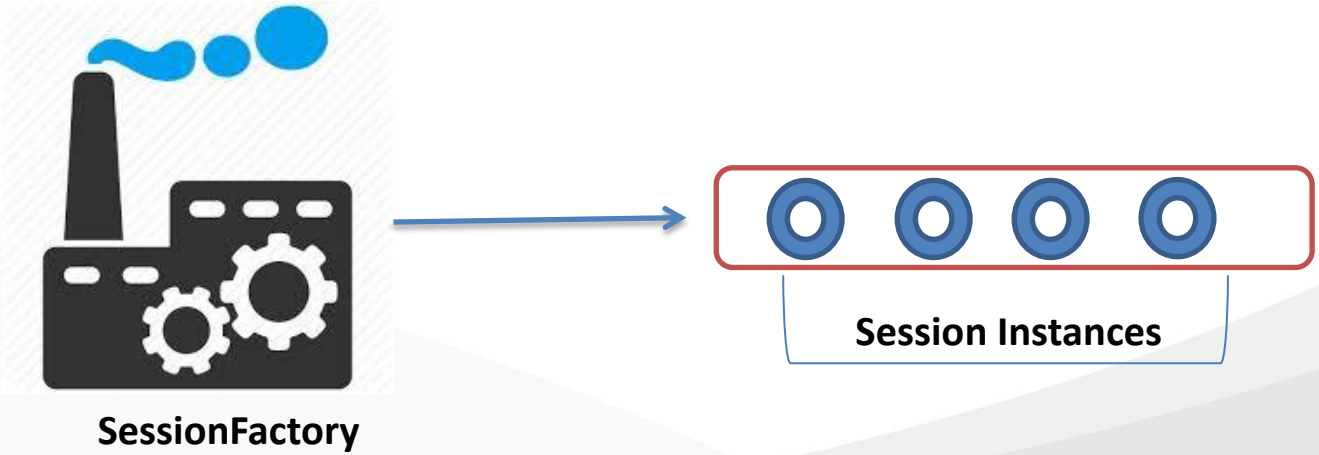
- SessionFactory
- Session
- Transaction
- TransactionFactory
- ConnectionProvider
- Persistent objects
- Transient objects

Agenda

- Introduction to ORM
- What is Hibernate & Why Hibernate?
- Hibernate Architecture
- **Hibernate Framework Objects**
- Instance Life Cycle

Session Factory

- `SessionFactory (org.hibernate.SessionFactory)`
  - It is a factory of session instances and client of ConnectionProvider
  - It is an expensive object to create. It is created during application start up
  - It is created once and closed at the end of the application lifecycle
  - It holds second level cache (optional) of data
  - Generally, an application has single SessionFactory and can be shared by all the application threads
  - The SessionFactory is created using configuration settings provided in the `hibernate.cfg.xml file`



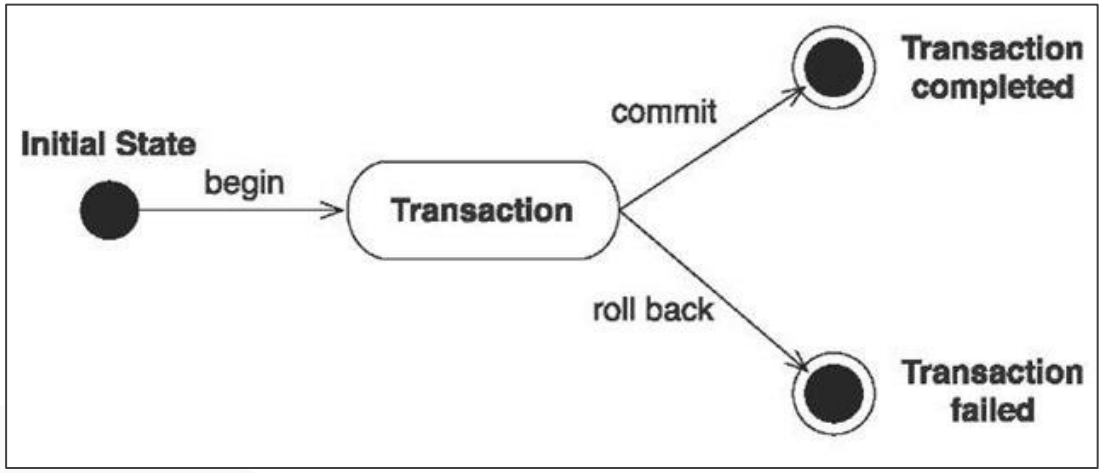
Session

- `Session (org.hibernate.Session)`
  - A session provides an interface between the application and data stored in the database
  - A session wraps a JDBC connection
  - The session instances are light weighted and can be created and destroyed without expensive process
  - Hibernate Session object represents a single unit-of-work for a given data store and is opened by a SessionFactory instance
  - Java objects is stored in database using Session object
  - It holds a first-level cache (mandatory) of data

```
Session session = factory.openSession();
```

Transaction (1/2)

- `Transaction (org.hibernate.Transaction)`
  - The Transaction object specifies the atomic unit of work
  - Transaction abstracts the application from the underlying JDBC or JTA transaction
  - A session might span several transactions
  - Transaction demarcation, either using the underlying API or Transaction object, is never optional
  - Following figure shows the unit-of-work managed by transaction



Source : https://www.helicaltech.com/transaction-in-hibernate/

Transaction (2/2)

- Following code snippet shows how to apply transaction boundary to a session:

```
Session sess = factory.openSession();
 Transaction tx = null;
 try {
       tx = sess.beginTransaction();
              //do some work ...
             tx.commit();
             tx=null;
    }
  catch (Exception e) {
         if (tx!=null)
              tx.rollback();
               throw e;
  }
  finally { sess.close();}
```

Transaction Factory

- `TransactionFactory (org.hibernate.TransactionFactory)`
  - It is a factory for Transaction instances
  - It is not exposed to the application, but can be extended/implemented by the developer
  - Example: `org.hibernate.transaction.JTATransactionFactory`

Connection Provider

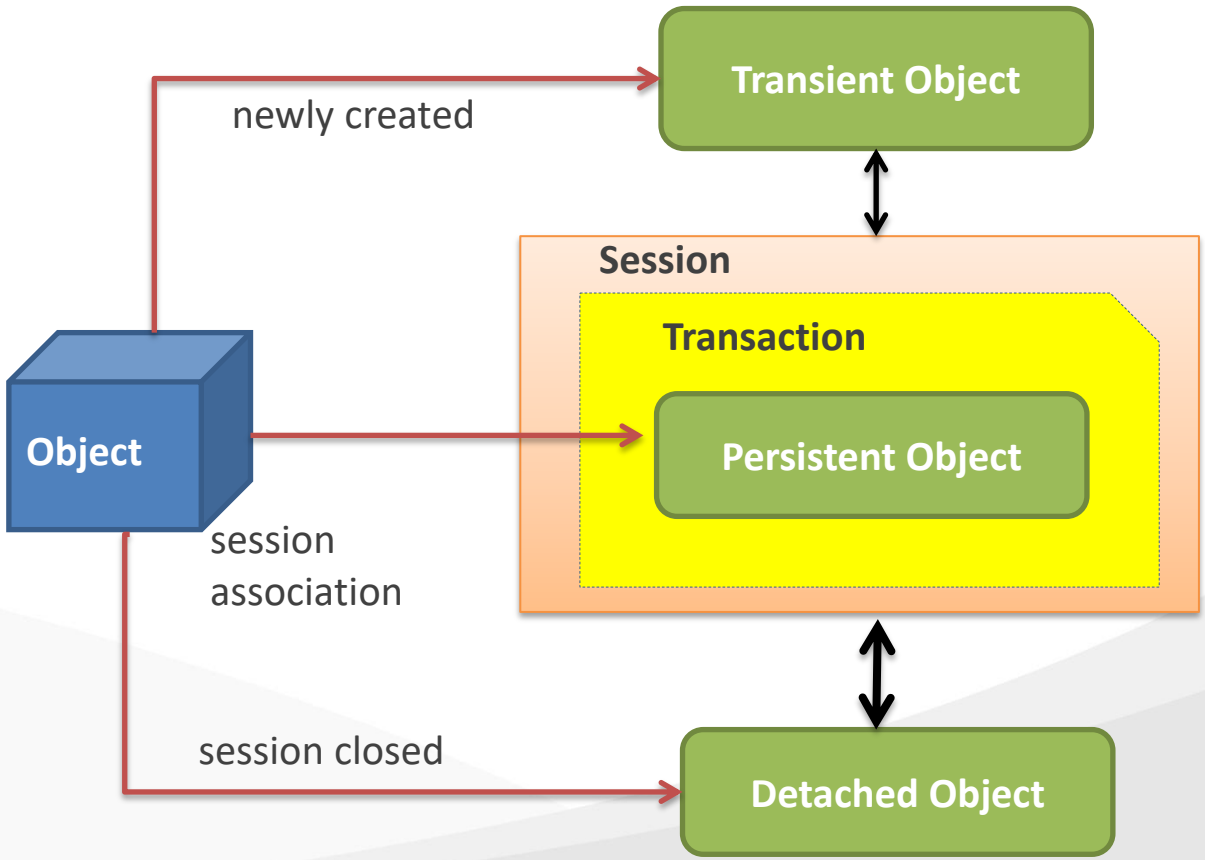
- `ConnectionProvider (org.hibernate.connection.ConnectionProvider)`
  - It is a factory for (and pool of) JDBC connections
  - It abstracts application from underlying Datasource or DriverManager
  - It is not exposed to application, but can be extended/implemented by the developer

Agenda

- Introduction to ORM
- What is Hibernate & Why Hibernate?
- Hibernate Architecture
- Hibernate Framework Objects
- **Instance Life Cycle**

Instance State in Hibernate Application Lifecycle

- An instance of a persistent classes may be in one of three different states, which are defined with respect to a persistence context
- Instances can be in one of the following states at a point of time:
  - Transient
  - Persistent
  - Detached



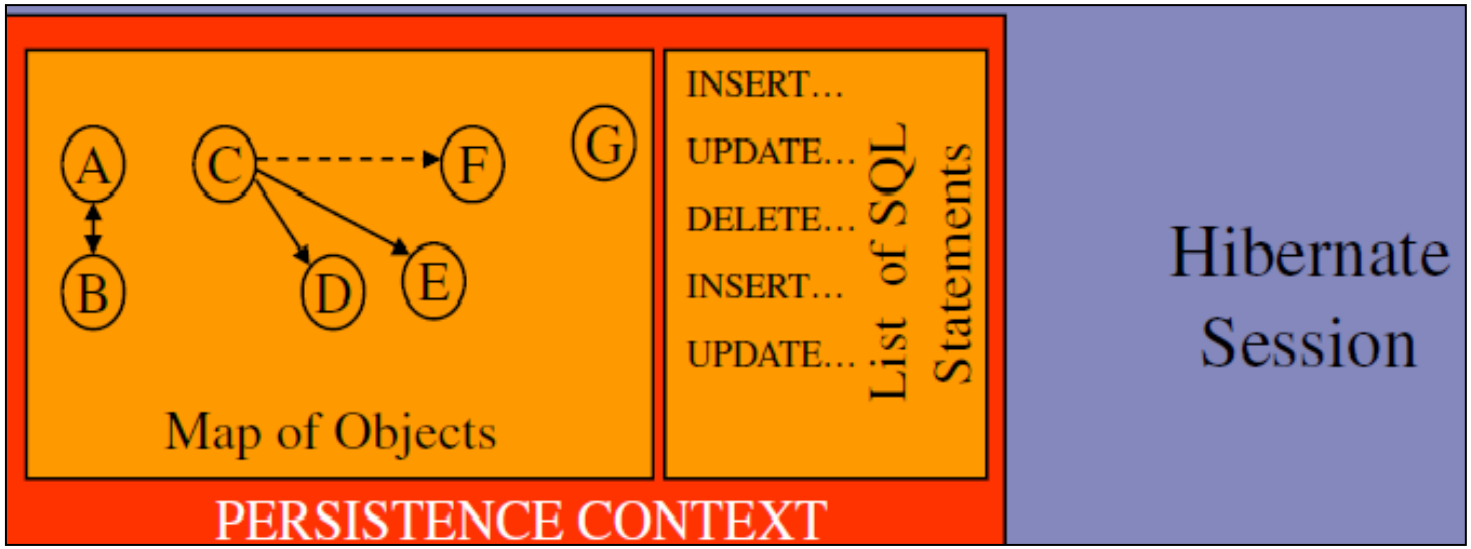
Persistent and Transient Objects

- Persistent objects:
  - Short-lived, single threaded objects containing persistent state and business function
  - They are associated with exactly one Session
- Persistence in Java
  - Instances of persistent classes that are not currently associated with a Session, thus without a persistent context
  - They may have been instantiated by the application and not (yet) persisted or they may have been instantiated by a closed Session
  - Changes made to Transient and detached objects do not get reflected to the database table

Objects that can be persisted in the database through hibernate framework can be ordinary JavaBeans/POJO

Instance States

- Object transitions from one state to other through various method calls
- The persistence context is represented by Hibernate Session object



Transient State

- When POJO instance is created it is in the transient state

```
Session session = factory.openSession();
```

  - account instance - is an transient object
- The instance is not, and has never been associated with any session (persistence context)
- It has no persistent identity (primary key value)
- It has no corresponding row in the database
- Also, when POJO instance is deleted from the session it moves from persistent to transient state again

```
session.delete(account);
```

- The Java object is still alive, though deleted from the database and stays alive until developer sets to null, or goes out of scope

Persistent State

- The instance is currently associated with a session (persistence context)
- It has a persistent identity (primary key value) and a corresponding row in the database
- Object moves from transient to persistent state when the POJO instance is persisted
- Any modification to an object in persistent state is automatically reflected to database (at transaction commit time)

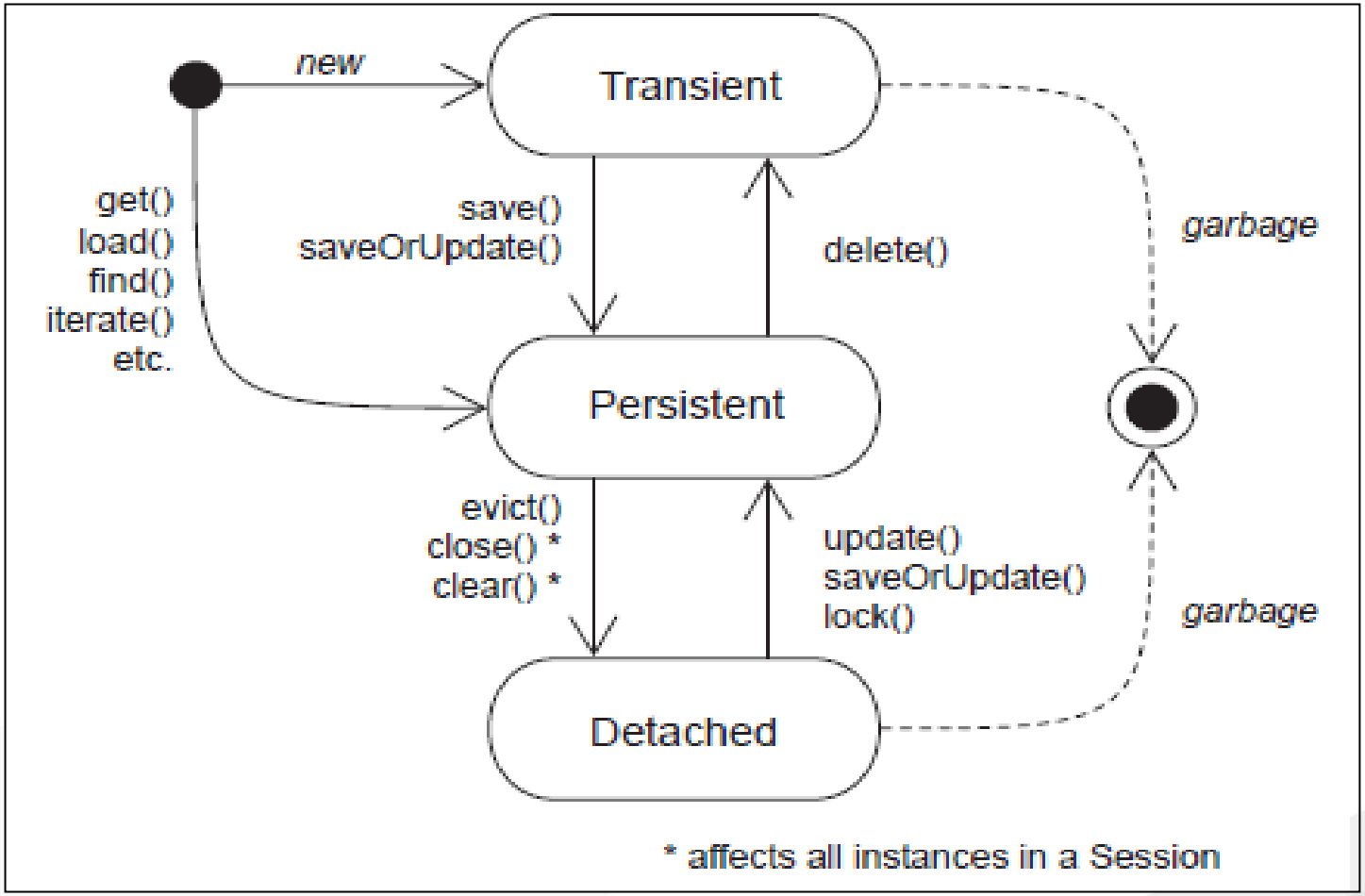
```
session.saveOrUpdate(account);
```

Detached State

- The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process
- It has a persistent identity and, perhaps, a corresponding row in the database
- An object is moved from the persistent to detached state by closing the session associated with it
- Used when POJO object instance needs to be sent over to another program for manipulation without having persistent context

```
Account account = session.get(Account.class, 1);
session1.close()
```

State Transitions (1/2)



Source : http://learnertobeginner.blogspot.com/2010/07/hibernate-persistence-lifecycle.html

State Transitions (2/2)

- Transient instances may be made persistent by calling `save()`, `persist()` or `saveOrUpdate()`
- Persistent instances may be made transient (i.e. in removed state) by calling `delete()`
- Any instance returned by a `get()` or `load()` method is persistent
- Detached instances may be made persistent by calling update(), saveOrUpdate(), `lock()` or `replicate()`
- The state of a transient or detached instance may also be made persistent as a new persistent instance by calling `merge()`

Thank You

CitiusTech Markets

CitiusTech Services

CitiusTech Platforms

Accelerating Innovation

CitiusTech Contacts

Email univerct@citiustech.com

www.citiustech.com