Spring Data

Kien Tran
Principal Software Engineer
11/11/2015

Client Logo

Introduction

- Your role
- Your background and experience in the subject
- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
- perform objective 1
- perform objective 2

Agenda

Course Audience and Prerequisite

- The course is for <whom>
- The following are prerequisites to <course>:
  - <knowledge>
  - <experiences>
  - <course>
  - …

Assessment Disciplines

- Class Participation: <%>
- Assignment: <%>
- Final Exam: <%>
- Passing Scores: <%>

Duration and Course Timetable

- Course Duration: <hrs>
- Course Timetable:
  - From <time> to <time>
  - Break <x> minutes from <time> to <time>

Further References

- <Source 1>
- <Source 2>
- …

Set Up Environment

- To complete the course, your PC must install:
  - Software 1
  - Software 2
  - …

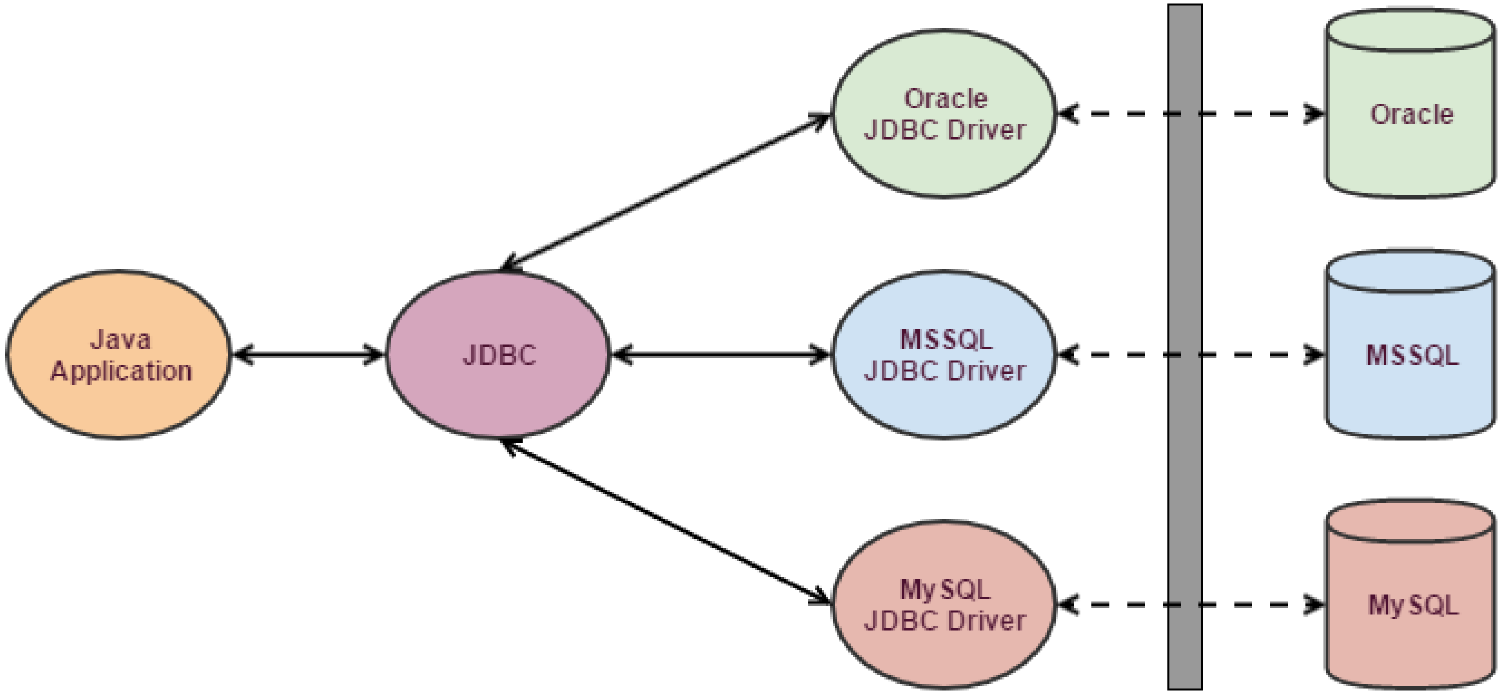
Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

JDBC

What is JDBC?

- JDBC (Java Database Connectivity) is a java API to connect and execute query with the database.
- JDBC API uses JDBC drivers to connect with the Relational Database.



Core JDBC Components

- DriverManager
  - This class manages a list of database drivers
- Connection
  - It uses a username, password, and a JDBC URL to establish a connection to the database and returns a connection object
  - A JDBC Connection represents a session/connection with a specific database

Core JDBC Components

- Statement
  - Use to execute queries and updates against the database.
- ResultSet
  - Perform a query against the database to get back a ResultSet. Then traverse this ResultSet to read the result of the query.
- SQLException
  - This class handles any errors that occur in a database application.

Kinds of Statements

- Statement
  - Execute simple SQL queries without parameters
- Prepared Statement
  - Execute precompiled SQL queries with or without parameters
- Callable Statement
  - Execute a call to a database stored procedure

JDBC: Work With MySQL

- Loading the JDBC Driver

```
Class.forName("com.mysql.jdbc.Driver");
```

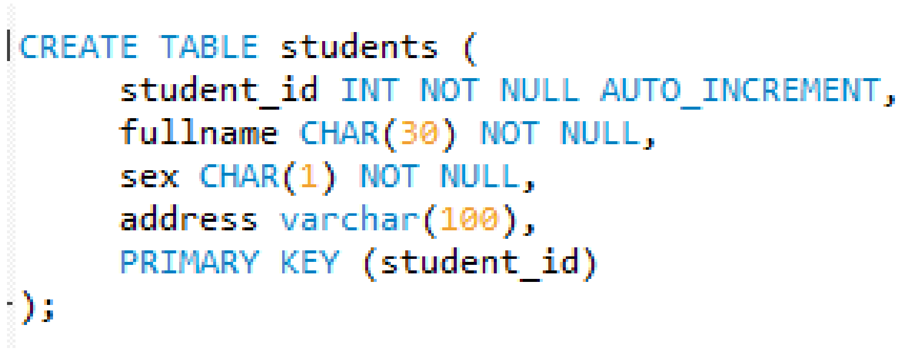
- Opening the Connection

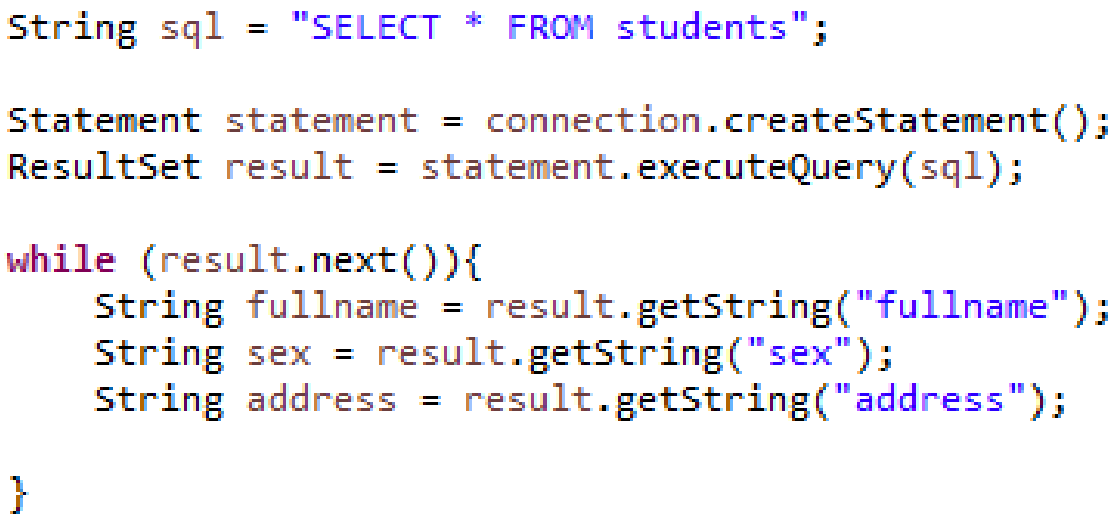
```
Connection conn = null;
conn = DriverManager.getConnection(
            "jdbc:mysql://hostname:port/dbname",
            "username",
            "password");
```

- Closing the Connection

```
conn.close();
```

Query the Database


```
CREATE TABLE students (
      student_id INT NOT NULL AUTO_INCREMENT,
      fullname CHAR(30) NOT NULL,
      sex CHAR(1) NOT NULL,
      address varchar(100),
      PRIMARY KEY (student_id)
);
```


```
String sql = "SELECT * FROM students";

Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery(sql);

while (result.next()){
    String fullname = result.getString("fullname");
    String sex = result.getString("sex");
    String address = result.getString("address");

}
```

Update the Database

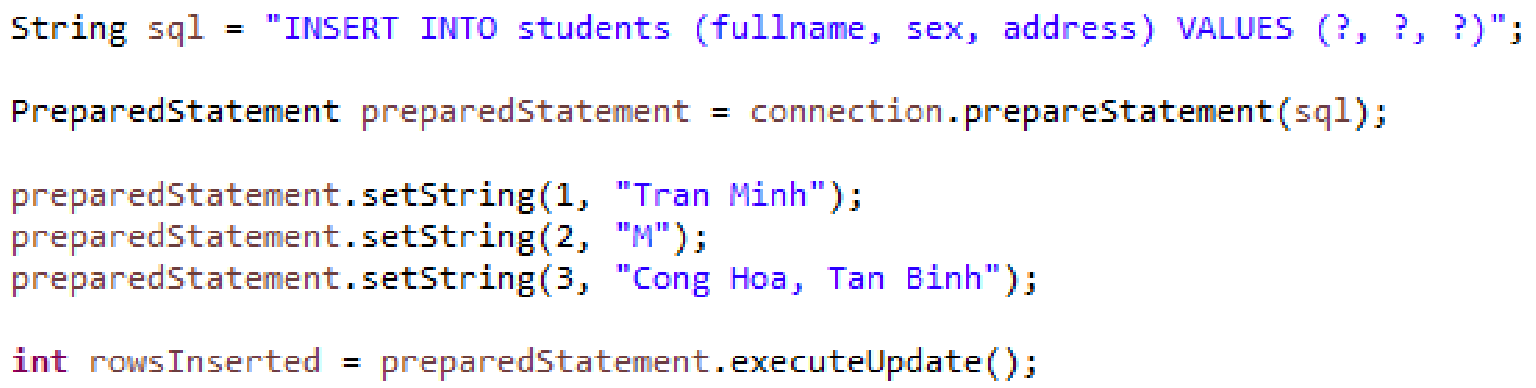
- Update records

```
String sql = "UPDATE students SET fullname = 'Nguyen Van A' WHERE student_id = 1";

Statement statement = null;
statement = connection.createStatement();
int rowAffected = statement.executeUpdate(sql);
```

- Delete records

```
String sql = "DELETE FROM students WHERE student_id = 1";

Statement statement = null;
statement = connection.createStatement();
int rowAffected = statement.executeUpdate(sql);
```

Using PreparedStatement

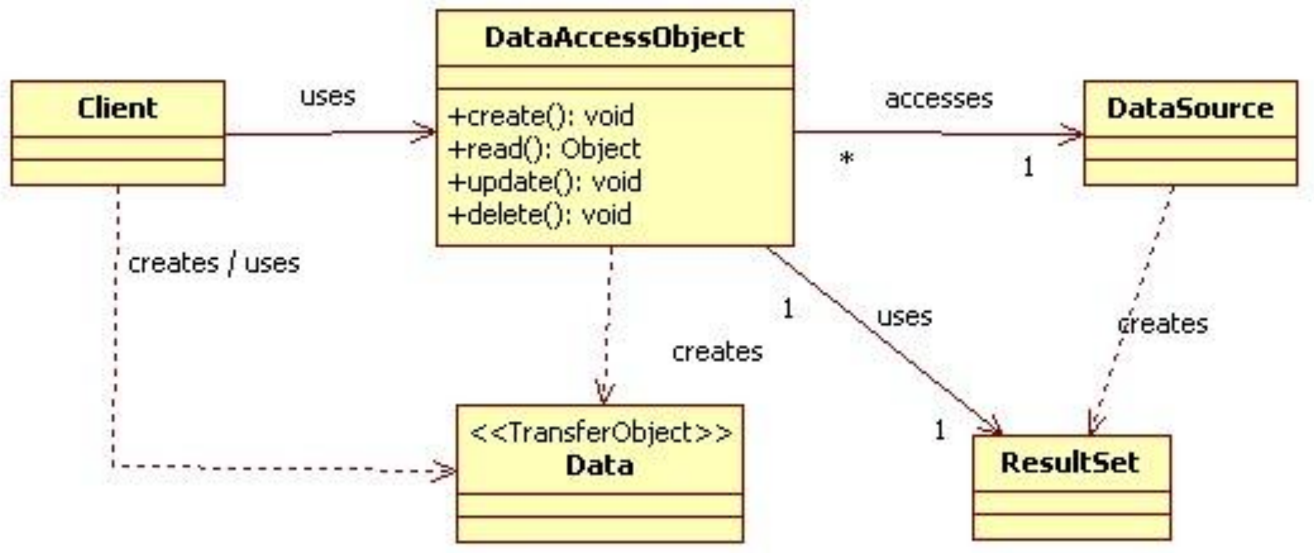


```
String sql = "INSERT INTO students (fullname, sex, address) VALUES (?, ?, ?)";

PreparedStatement preparedStatement = connection.prepareStatement(sql);

preparedStatement.setString(1, "Tran Minh");
preparedStatement.setString(2, "M");
preparedStatement.setString(3, "Cong Hoa, Tan Binh");

int rowsInserted = preparedStatement.executeUpdate();
```

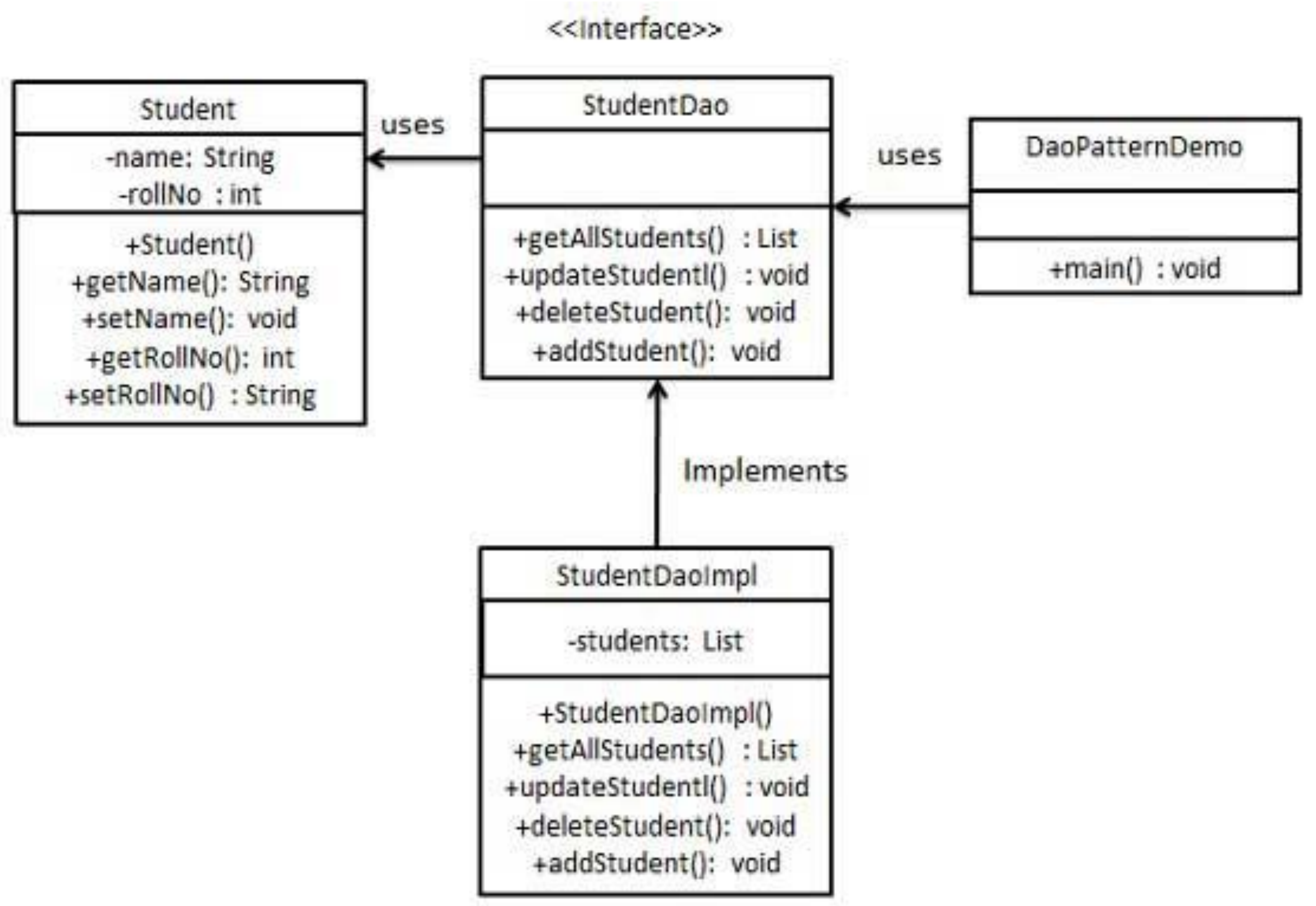
Data Access Object

Data Access Object Pattern

- DAO pattern is used to separate low level data accessing API or operations from high level business services



Sample



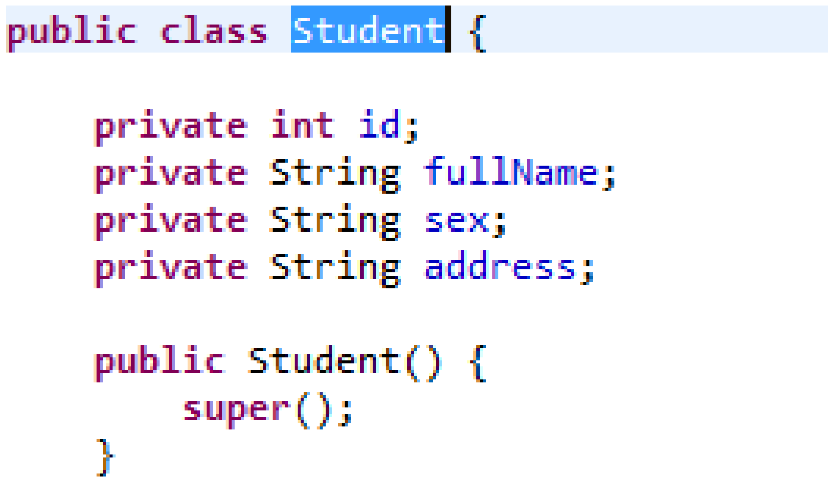
<<Interface>>
Student
-name: String
-rollNo : int
+Student()
+getName(): String
+setName(): void
+getRollNo(): int
+setRollNo() : String
uses
StudentDao
+getAllStudents() : List
+updateStudentl() : void
+deleteStudent(): void
+addStudent(): void
uses
DaoPatternDemo
+main() : void
Implements
StudentDaoImpl
-students: List
+StudentDaoImpl()
+getAllStudents() : List
+updateStudentl() : void
+deleteStudent(): void
+addStudent(): void

The participants in Data Access Object Pattern

- Data Access Object Interface
  - This interface defines the standard operations to be performed on a model object(s)
- Data Access Object concrete class
  - This class implements above interface.
  - This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- Model Object or Value Object
  - This object is simple POJO containing get/set methods to store data retrieved using DAO class

Implementation

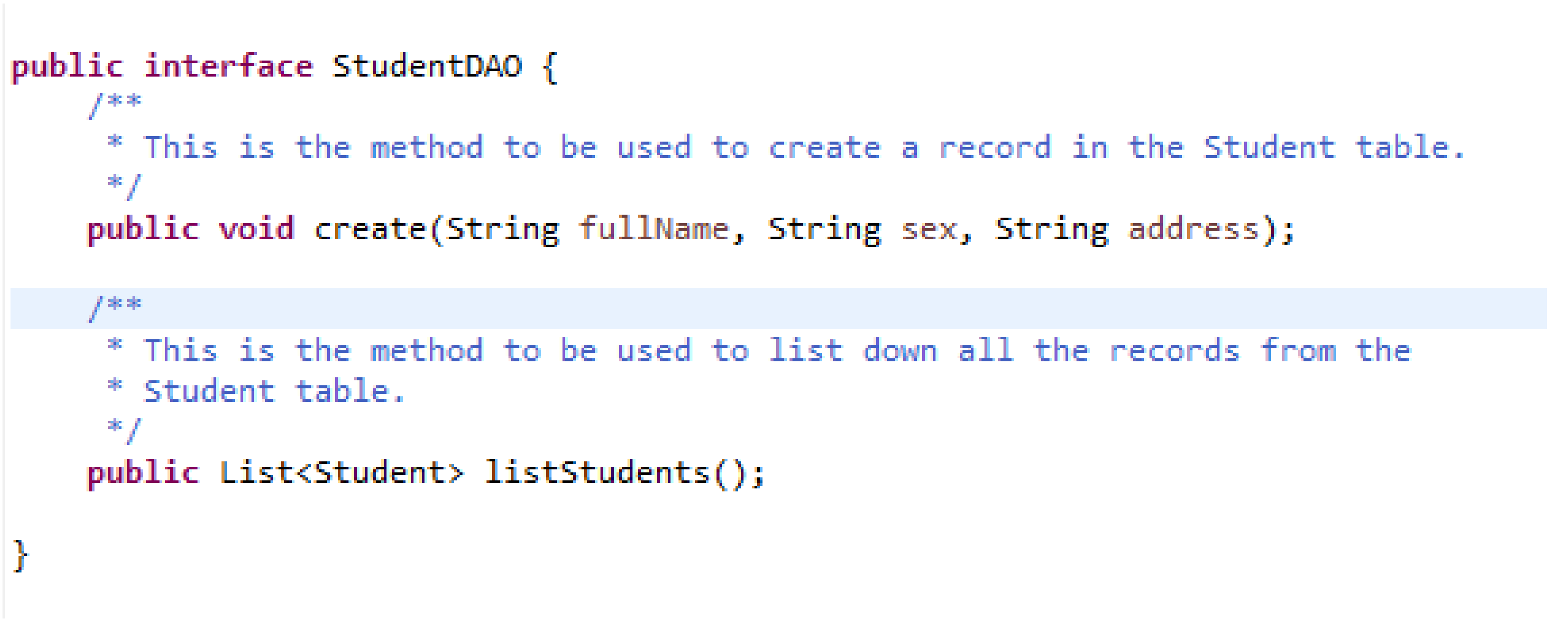
- Student



```
public class Student {

    private int id;
    private String fullName;
    private String sex;
    private String address;

    public Student() {
        super();
    }
```

Implementation

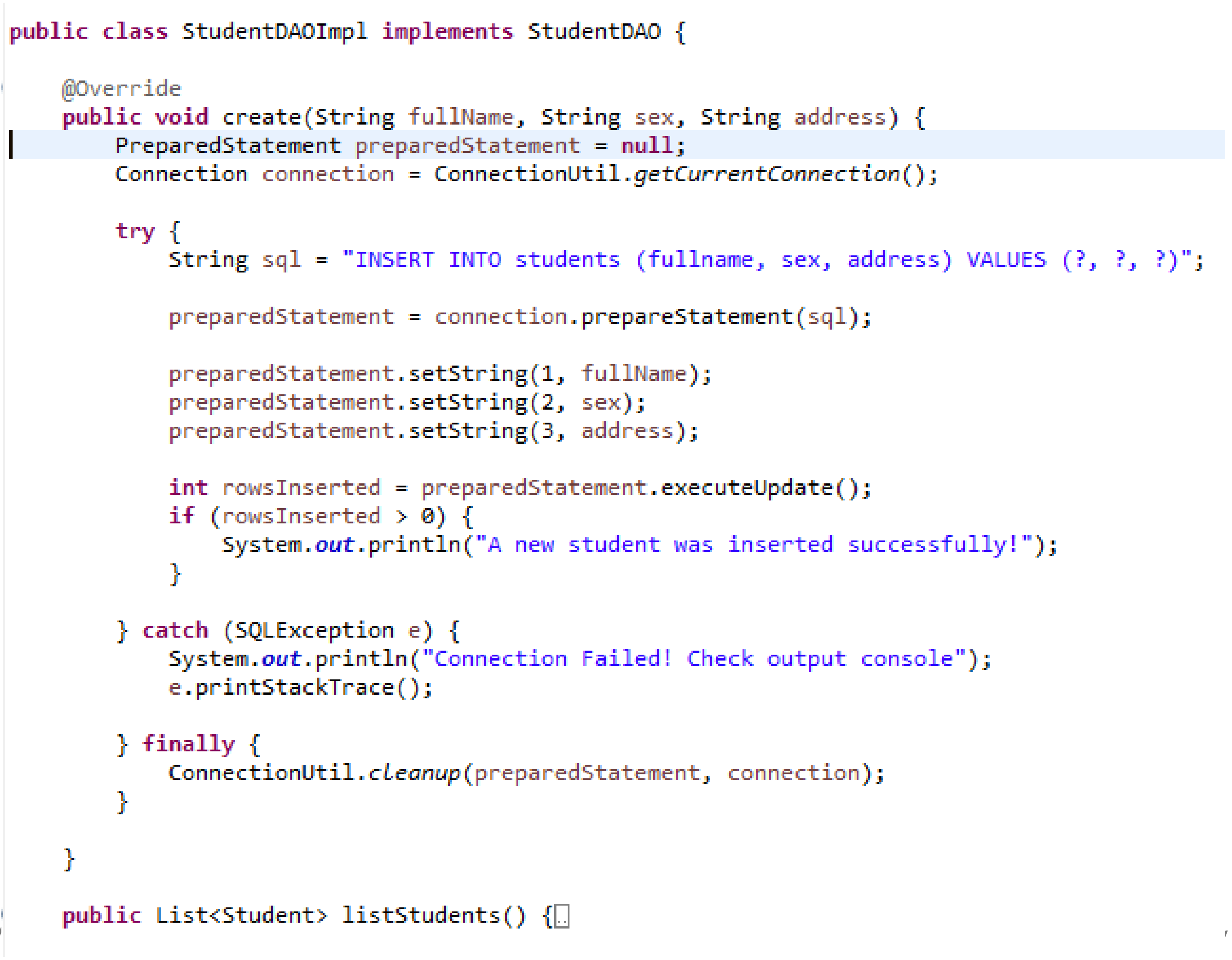
- StudentDAO



```
public interface StudentDAO {
    /**
     * This is the method to be used to create a record in the Student table.
     */
    public void create(String fullName, String sex, String address);

    /**
     * This is the method to be used to list down all the records from the
     * Student table.
     */
    public List<Student> listStudents();

}
```

Implementation

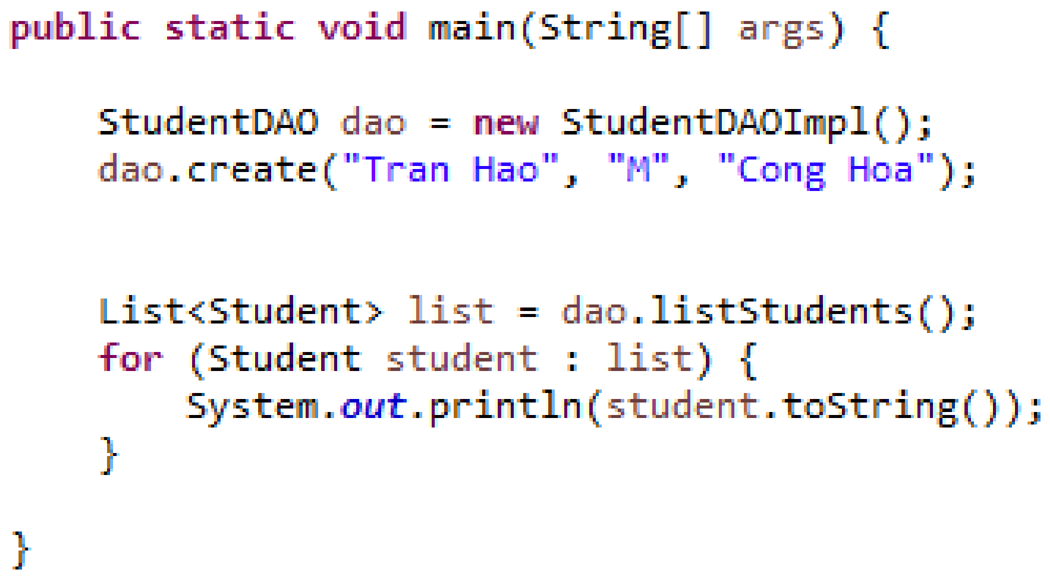
- StudentDAOImpl



```
public class StudentDAOImpl implements StudentDAO {

    @Override
    public void create(String fullName, String sex, String address) {
        PreparedStatement preparedStatement = null;
        Connection connection = ConnectionUtil.getCurrentConnection();

        try {
            String sql = "INSERT INTO students (fullname, sex, address) VALUES (?, ?, ?)";

            preparedStatement = connection.prepareStatement(sql);

            preparedStatement.setString(1, fullName);
            preparedStatement.setString(2, sex);
            preparedStatement.setString(3, address);

            int rowsInserted = preparedStatement.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("A new student was inserted successfully!");
            }

        } catch (SQLException e) {
            System.out.println("Connection Failed! Check output console");
            e.printStackTrace();

        } finally {
            ConnectionUtil.cleanup(preparedStatement, connection);
        }

    }

    public List<Student> listStudents() {
```

Implementation

- StudentDAODemo

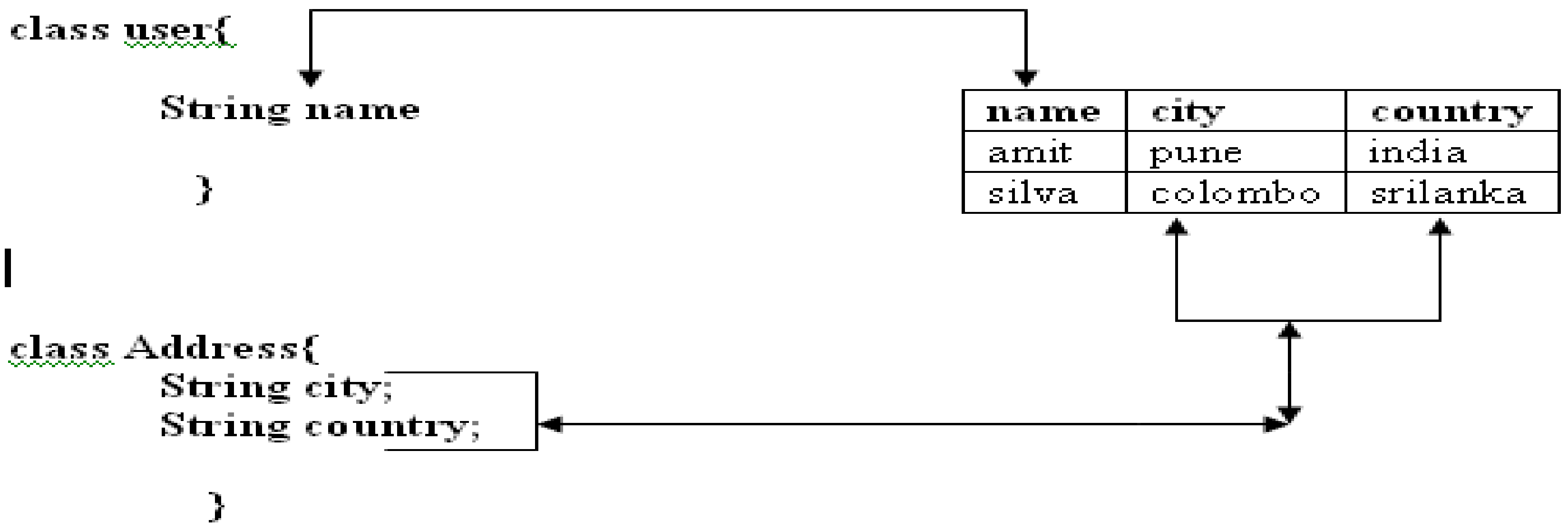


```java
public static void main(String[] args) {

    StudentDAO dao = new StudentDAOImpl();
    dao.create("Tran Hao", "M", "Cong Hoa");


    List<Student> list = dao.listStudents();
    for (Student student : list) {
        System.out.println(student.toString());
    }

}
```

Hibernate

Object-Relation Impedence Mismatch

- Granularity
  - Sometimes you will have an object model, which has more classes than the number of corresponding tables in the database (The object model is more granular than the relational model) and vice versa.

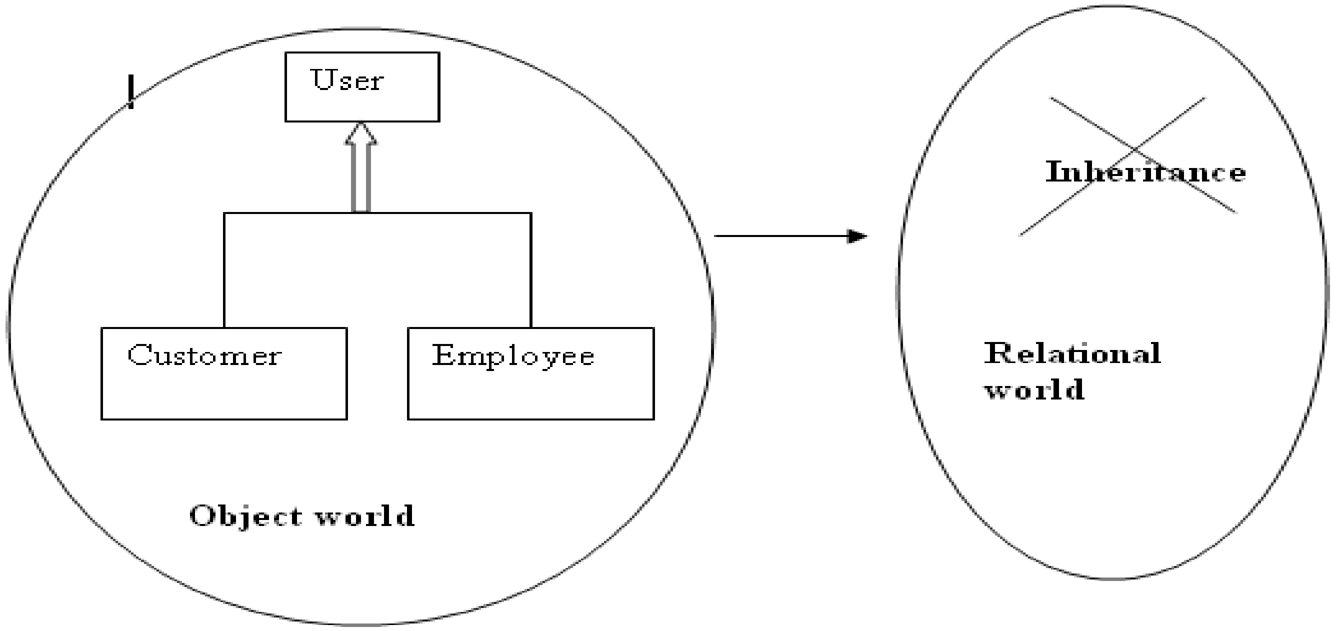


```
class user{
        String name
  }
|
class Address{
        String city;
        String country;
  }
```

name	city	country
amit	pune	india
silva	colombo	srilanka

Object-Relation Impedence Mismatch

- Subtypes (inheritance)
  - Inheritance is a natural paradigm in object-oriented programming languages. However, RDBMSs do not define anything similar on the whole.

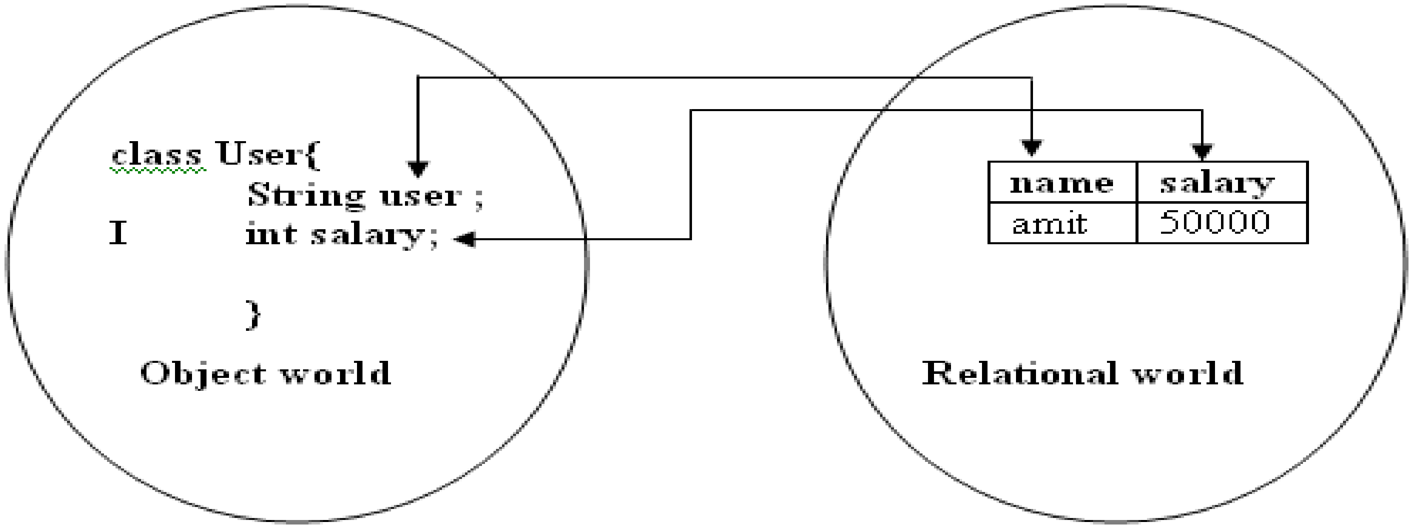


Object-Relation Impedence Mismatch

- Identity
  - An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).

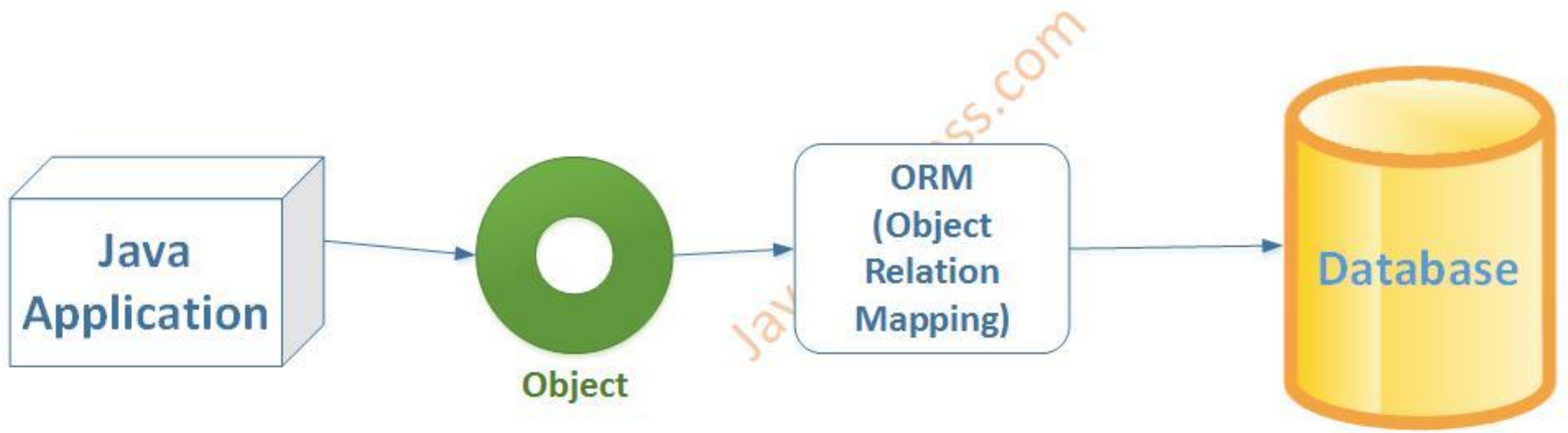
What is Object/Relation Mapping?

- It means how we will map the relational world with the object world.
  - In the relational world, the data is in the form of a table that contains rows and column.
  - In the object world, the data is in the form of an object.



What is Hibernate?

- Hibernate is one of the most popular Object/Relational Mapping (ORM) framework in the Java world
- It allows developers to map the object structures of normal Java classes to the relational structure of a database
- Hibernate framework simplifies the development of java application to interact with the database

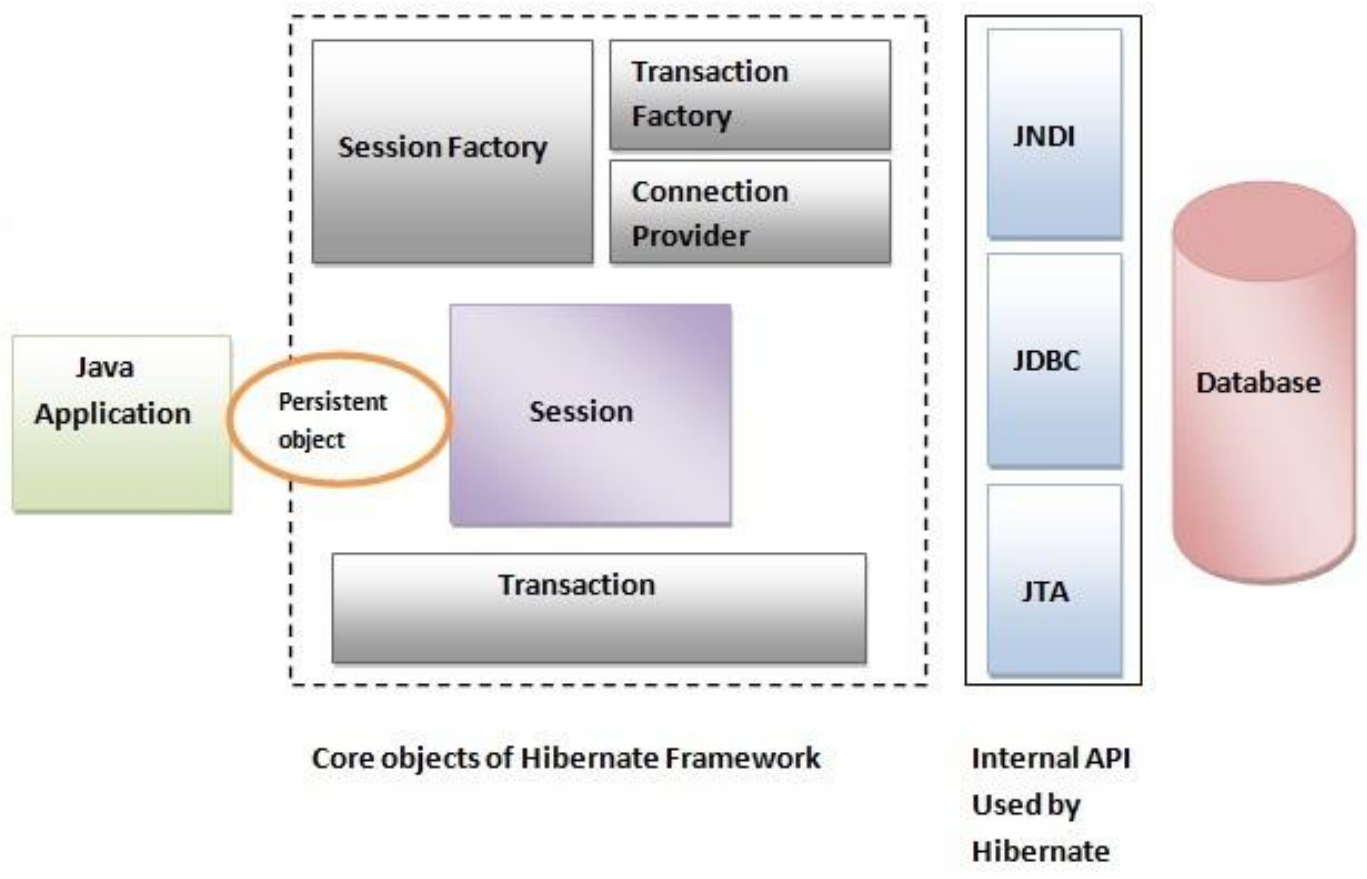


Hibernate Flow Diagram

Advantages of hibernates

- Supports Inheritance, Collections (List,Set,Map)
- Supports relationships like One-To-Many,One-To-One, Many-To-Many, Many-To-One
- HQL (Hibernate Query Language) - database independent commands
- Simplifies complex join
- Supports caching mechanism - Fast performance

Hibernate Architecture



Elements of Hibernate Architecture

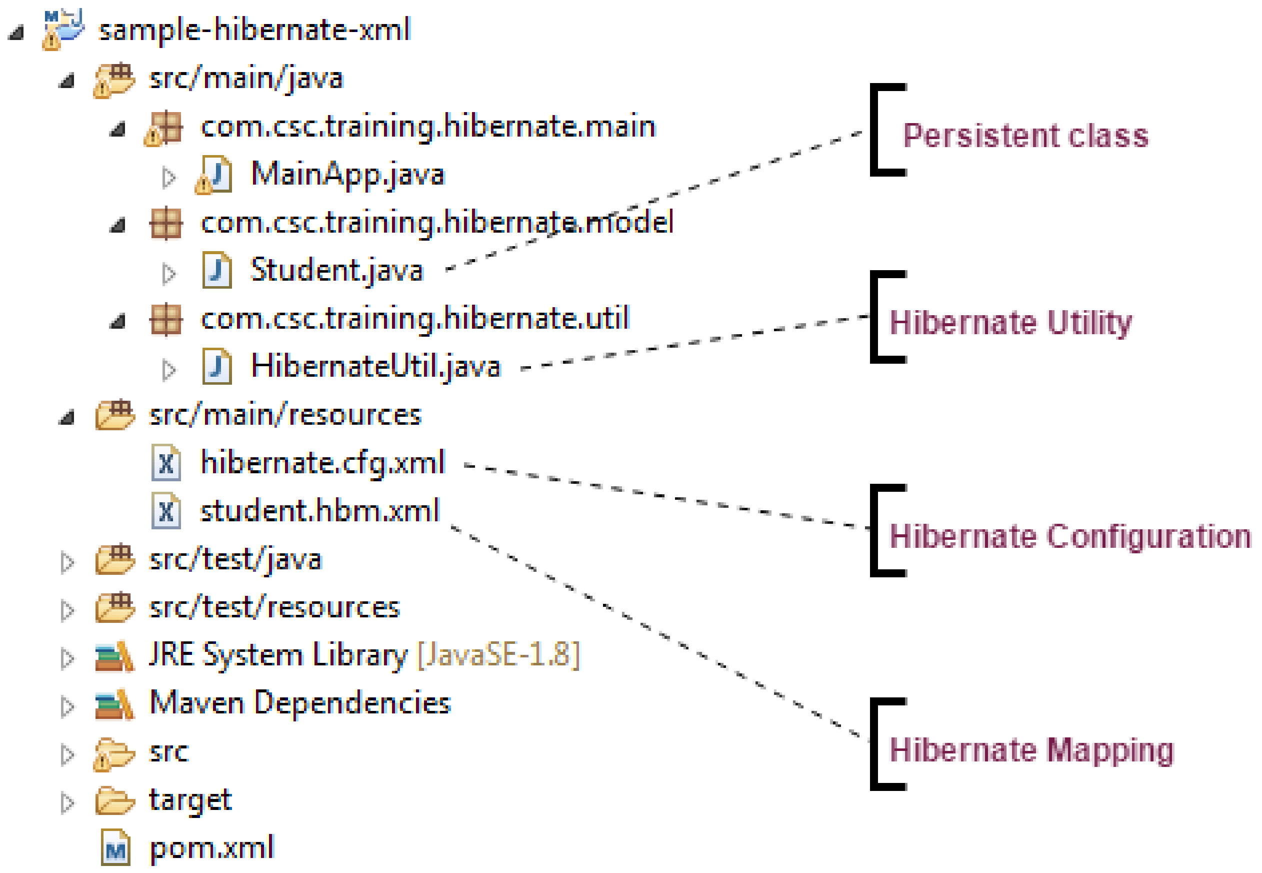
- SessionFactory
  - A factory for Session and a client of ConnectionProvider.
  - An interface provides factory method to get the object of Session

- Session
  - An interface between the application and data stored in the database
  - It is a short-lived object and wraps the JDBC connection
  - Provides methods to insert, update and delete the object
  - Provides factory methods for Transaction, Query and Criteria

Elements of Hibernate Architecture

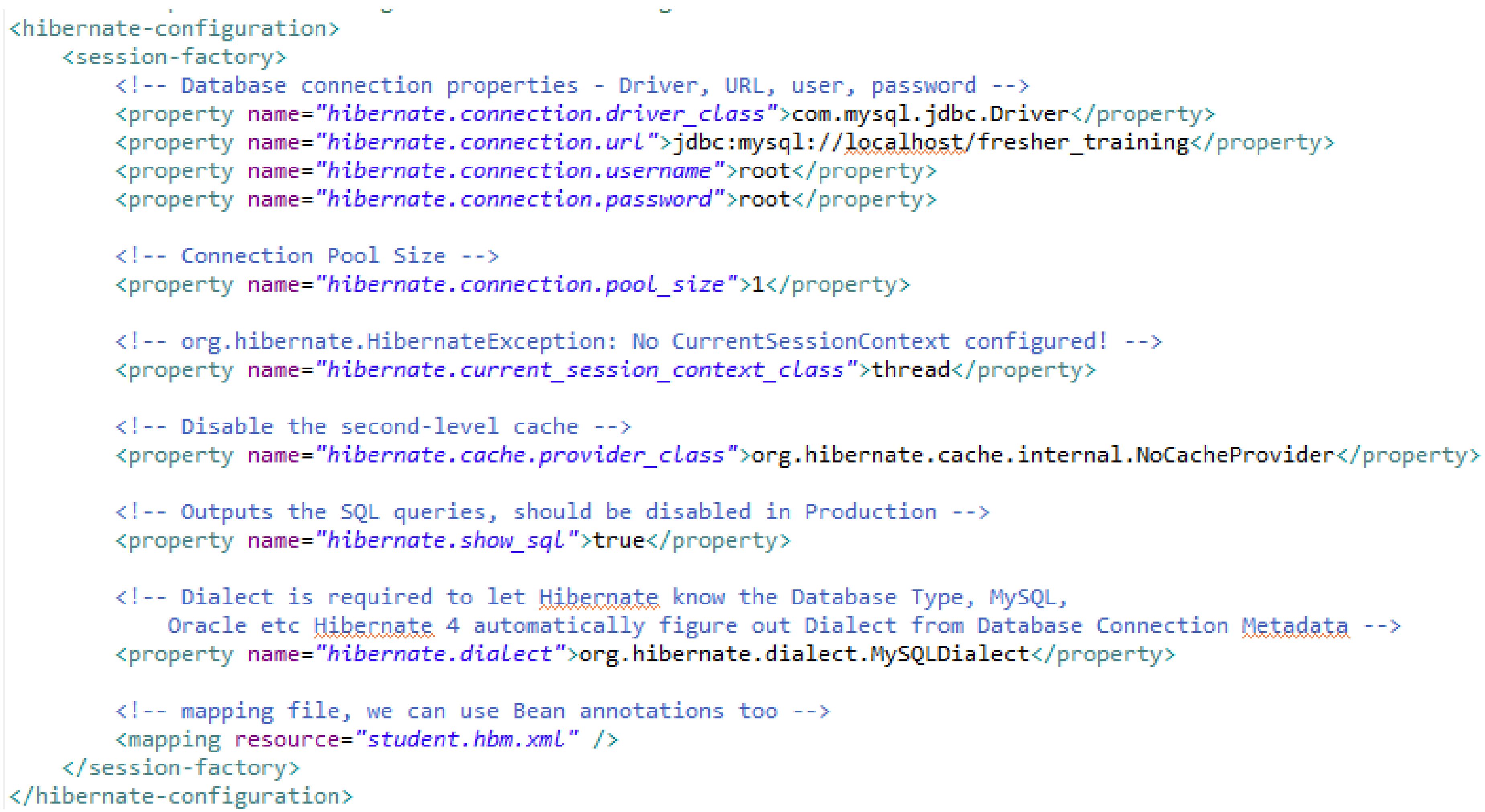
- Transaction
  - Provides methods for transaction management
- Query
  - Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects

Hibernate Sample



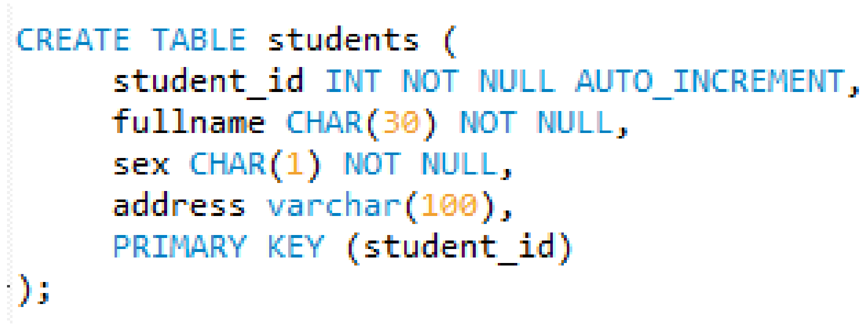
```
sample-hibernate-xml
  src/main/java
    com.csc.training.hibernate.main
      MainApp.java
    com.csc.training.hibernate.model
      Student.java
    com.csc.training.hibernate.util
      HibernateUtil.java
  src/main/resources
    hibernate.cfg.xml
    student.hbm.xml
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

Hibernate Configuration

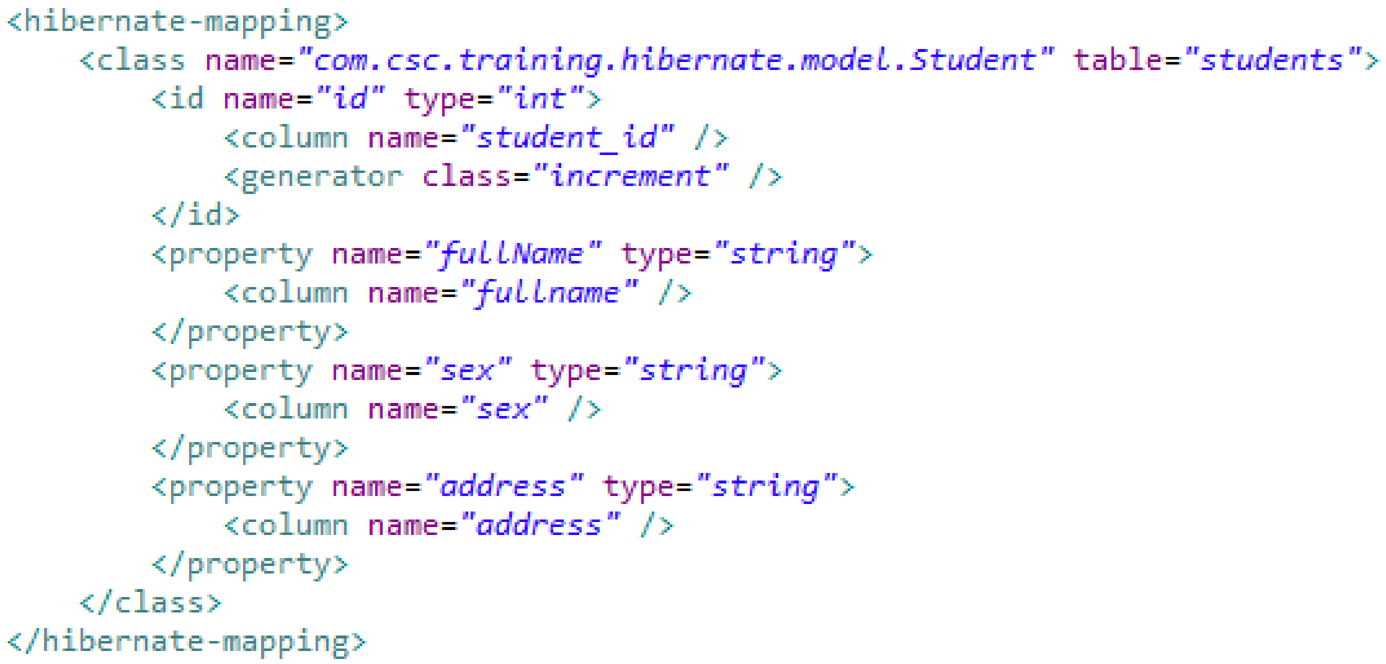


```
<hibernate-configuration>
    <session-factory>
        <!-- Database connection properties - Driver, URL, user, password -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost/fresher_training</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>

        <!-- Connection Pool Size -->
        <property name="hibernate.connection.pool_size">1</property>

        <!-- org.hibernate.HibernateException: No CurrentSessionContext configured! -->
        <property name="hibernate.current_session_context_class">thread</property>

        <!-- Disable the second-level cache -->
        <property name="hibernate.cache.provider_class">org.hibernate.cache.internal.NoCacheProvider</property>

        <!-- Outputs the SQL queries, should be disabled in Production -->
        <property name="hibernate.show_sql">true</property>

        <!-- Dialect is required to let Hibernate know the Database Type, MySQL,
            Oracle etc Hibernate 4 automatically figure out Dialect from Database Connection Metadata -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- mapping file, we can use Bean annotations too -->
        <mapping resource="student.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

Hibernate Mapping Files

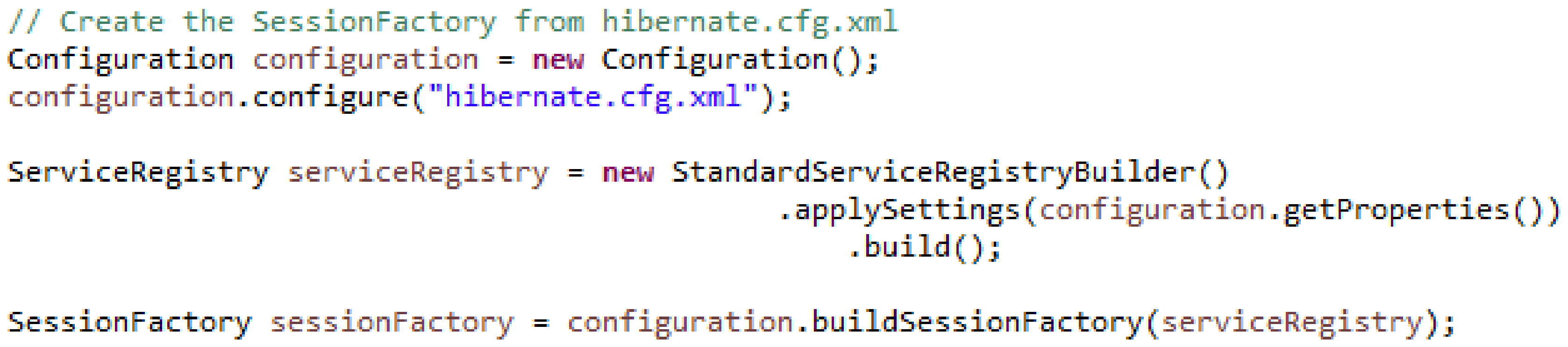


```sql
CREATE TABLE students (
      student_id INT NOT NULL AUTO_INCREMENT,
      fullname CHAR(30) NOT NULL,
      sex CHAR(1) NOT NULL,
      address varchar(100),
      PRIMARY KEY (student_id)
);
```



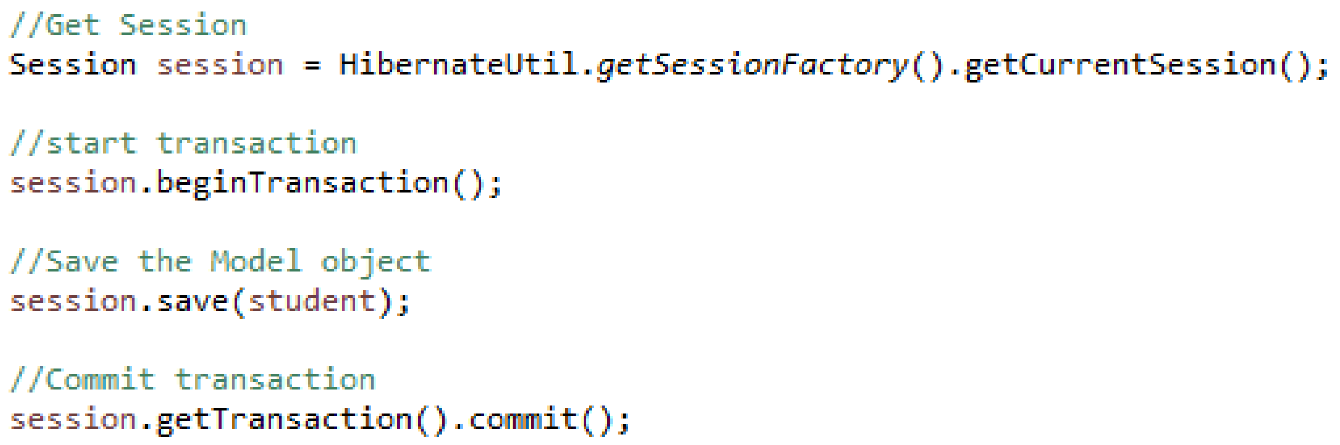
```xml
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.Student" table="students">
        <id name="id" type="int">
            <column name="student_id" />
            <generator class="increment" />
        </id>
        <property name="fullName" type="string">
            <column name="fullname" />
        </property>
        <property name="sex" type="string">
            <column name="sex" />
        </property>
        <property name="address" type="string">
            <column name="address" />
        </property>
    </class>
</hibernate-mapping>
```

Build SessionFactory



```java
// Create the SessionFactory from hibernate.cfg.xml
Configuration configuration = new Configuration();
configuration.configure("hibernate.cfg.xml");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                                            .applySettings(configuration.getProperties())
                                                .build();

SessionFactory sessionFactory = configuration.buildSessionFactory(serviceRegistry);
```

Using Hibernate to store Object to Database

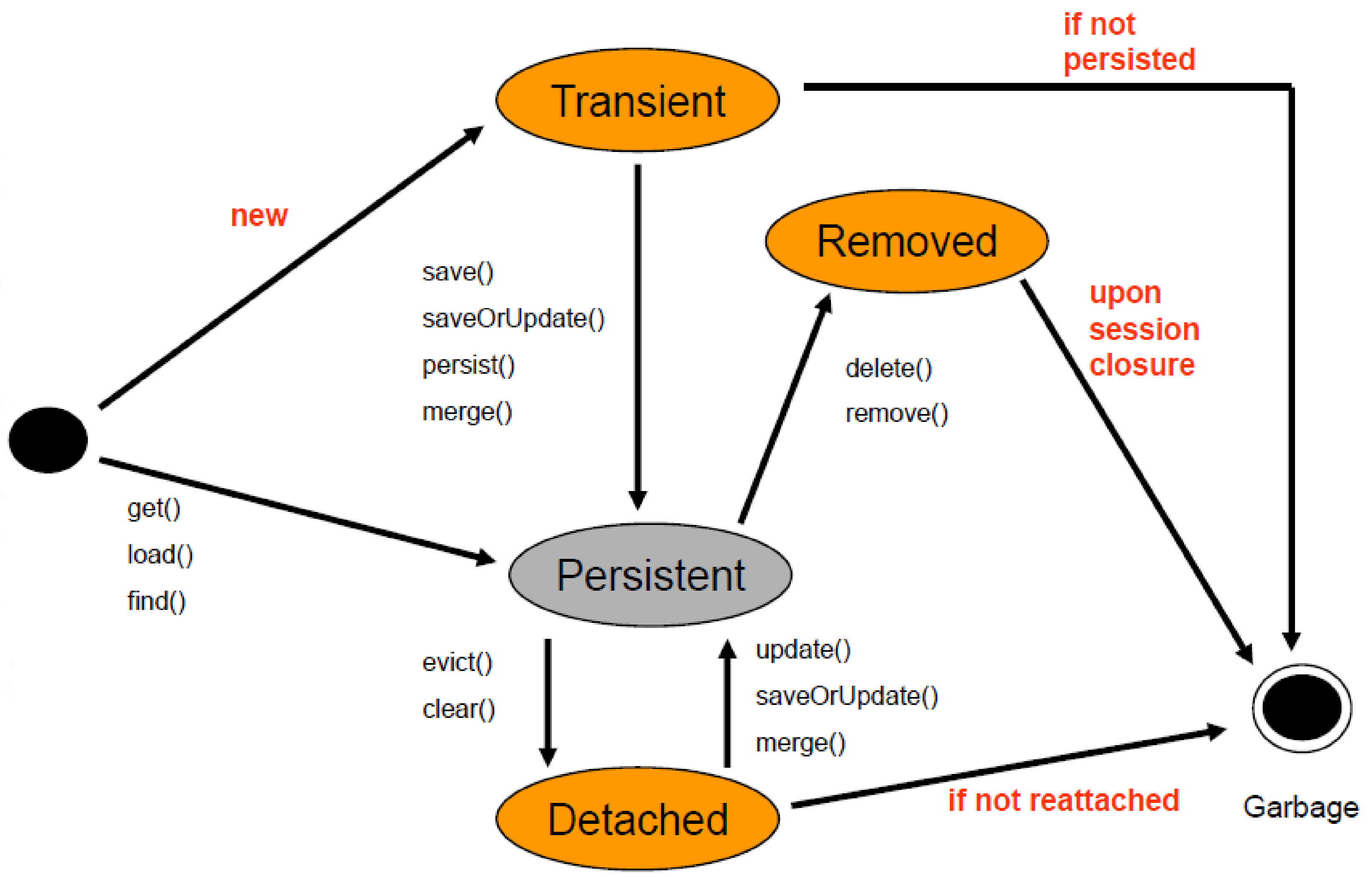


```java
//Get Session
Session session = HibernateUtil.getSessionFactory().getCurrentSession();

//start transaction
session.beginTransaction();

//Save the Model object
session.save(student);

//Commit transaction
session.getTransaction().commit();
```

Persistence Contexts

- **`org.hibernate.Session`** API represent a context for dealing with persistent data. This concept is called a persistence context.
- Persistent data has a state in relation to both a persistence context and the underlying database.

Entity states



Entity states

- transient
  - The entity has just been instantiated and is not associated with a persistence context.
  - It has no persistent representation in the database and no identifier value has been assigned.

- Persistent
  - the entity has an associated identifier and is associated with a persistence context.

Entity states

- detached
  - the entity has an associated identifier, but is no longer associated with a persistence context

- removed
  - the entity has an associated identifier and is associated with a persistence context, however it is scheduled for removal from the database.

Working with Entities

- Making entities persistent

```
DomesticCat fritz = new DomesticCat();
fritz.setColor(Color.GINGER);
fritz.setSex('M');
fritz.setName("Fritz");
Long generatedId = (Long) sess.save(fritz);

sess.persist(fritz);
```

- Loading an object

```
Cat fritz = (Cat) sess.load(Cat.class, generatedId);

Cat cat = (Cat) sess.get(Cat.class, id);
```

- Deleting entities

```
session.delete( fritz );
```

Hibernate Data Manipulation

- Hibernate Query Language (HQL)
  - The syntax is quite similar to database SQL language
  - HQL uses class name instead of table name, and property names instead of column name
- Hibernate Criteria
  - An alternative to Hibernate Query Language (HQL)
  - It's always a good solution in many optional search criteria
- Native SQL
  - Use the native database SQL language directly

HQL Query Sample

```
from Cat

from Formula, Parameter

select cat.name from DomesticCat cat where cat.name like 'fri%‘

select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat) from Cat
cat

from DomesticCat cat order by cat.name asc, cat.weight desc
```

Criteria Query Sample

```
Criteria crit = sess.createCriteria(Cat.class);
crit.setMaxResults(50); List cats = crit.list();

List cats = sess.createCriteria(Cat.class)
             .add( Restrictions.like("name", "Fritz%") )
             .add( Restrictions.between("weight", minWeight, maxWeight) )
             .list();

List cats = sess.createCriteria(Cat.class)
             .add( Restrictions.like("name", "F%")
             .addOrder( Order.desc("age") )
             .list();
```

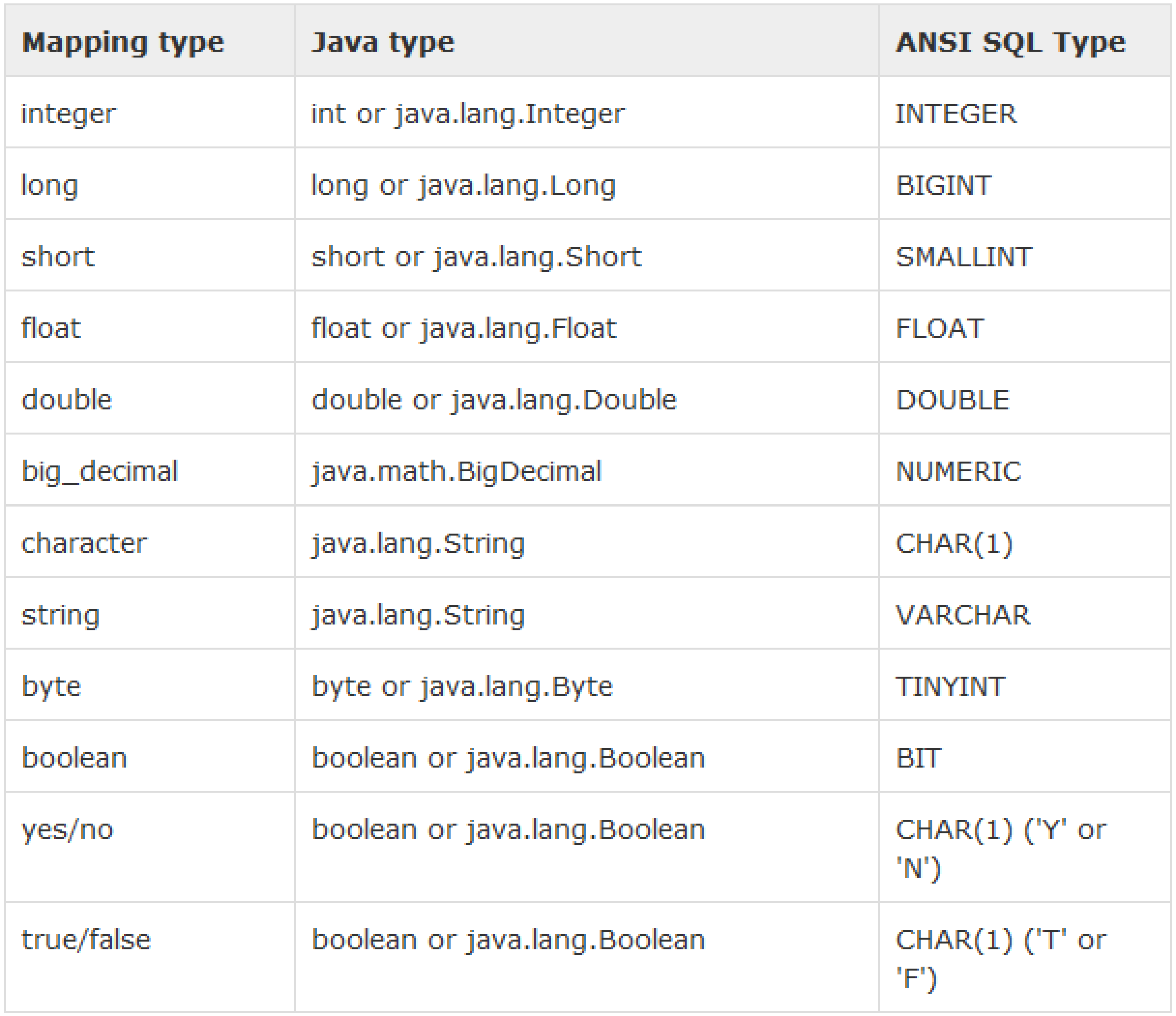
Native SQL Sample

```
sess.createSQLQuery("SELECT * FROM CATS").list();

sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS")
         .addEntity(Cat.class)
         .list();

sess.createSQLQuery("SELECT {cat.*}, {m.*} FROM CATS c, CATS m
                                              WHERE c.MOTHER_ID = m.ID")
        .addEntity("cat", Cat.class)
        .addEntity("mother", Cat.class)
        .list();
```

Hibernate mapping types

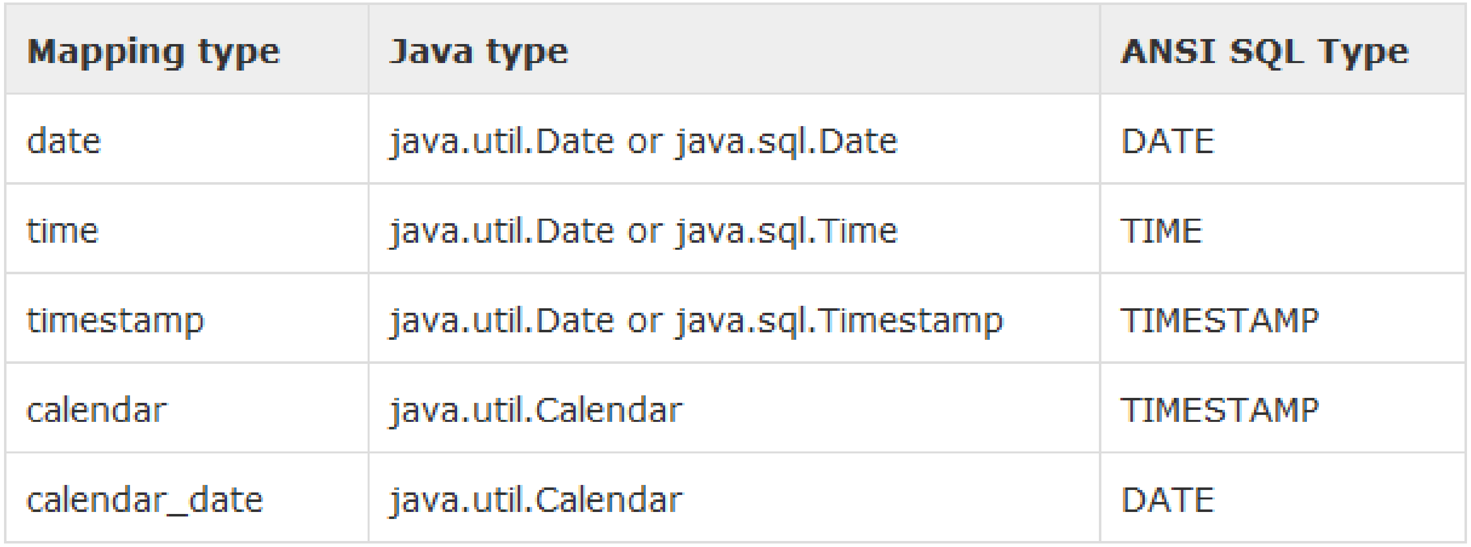
- The types declared and used in the mapping files are not Java data types; they are not SQL database types either
- Converters which can translate from Java to SQL data types and vice versa

Primitive types



Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

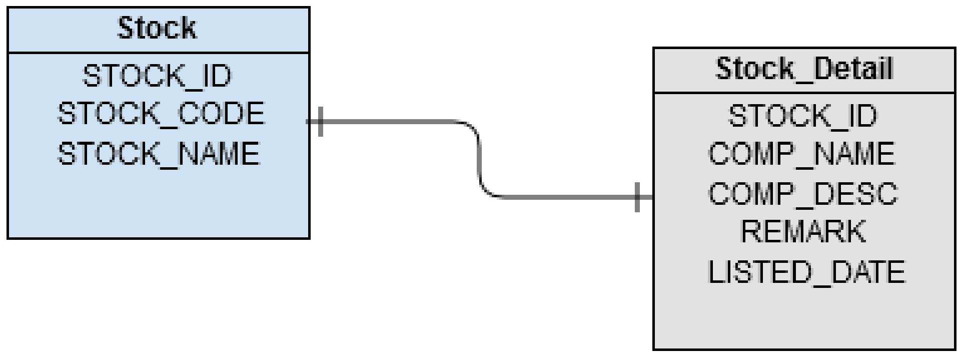
Date and time types



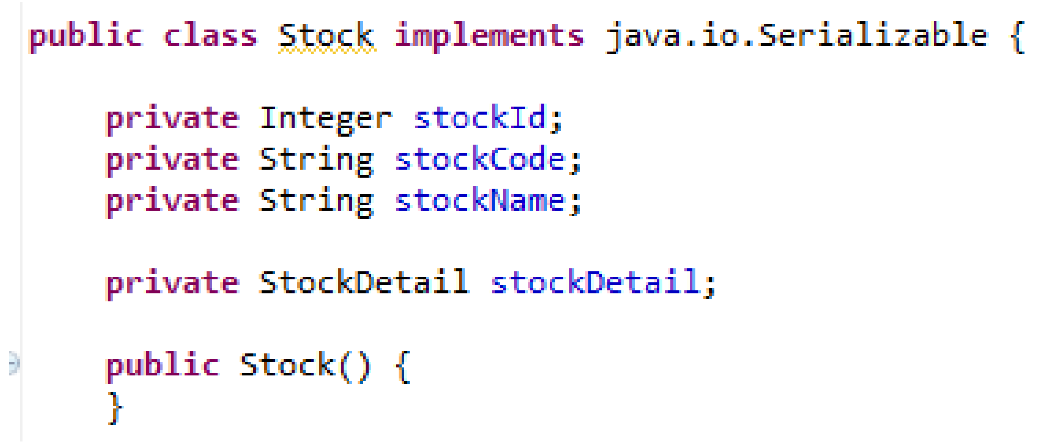
Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

One-to-One example (XML Mapping)

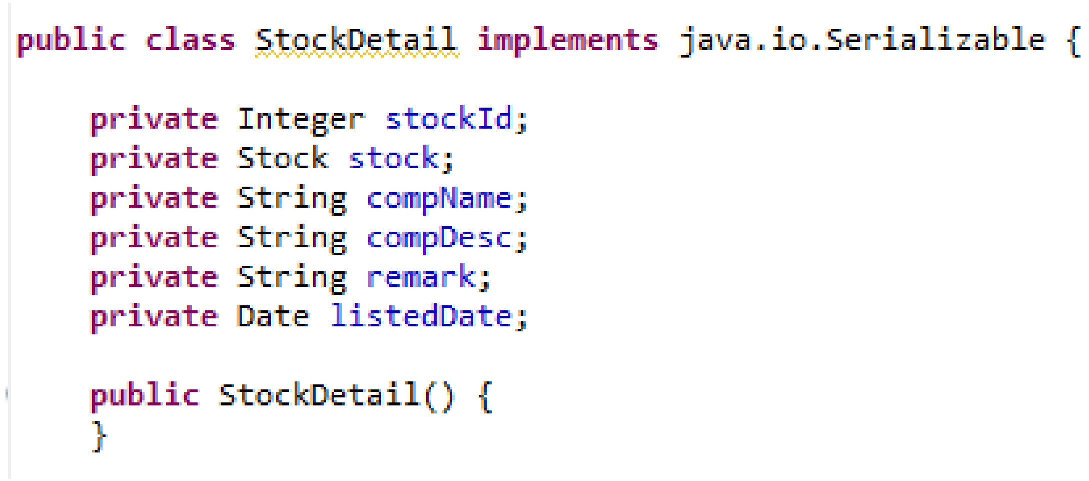
Entity Relationship Diagram (ERD)



One-to-One example (XML Mapping)



```java
public class Stock implements java.io.Serializable {

    private Integer stockId;
    private String stockCode;
    private String stockName;

    private StockDetail stockDetail;

    public Stock() {
    }
```

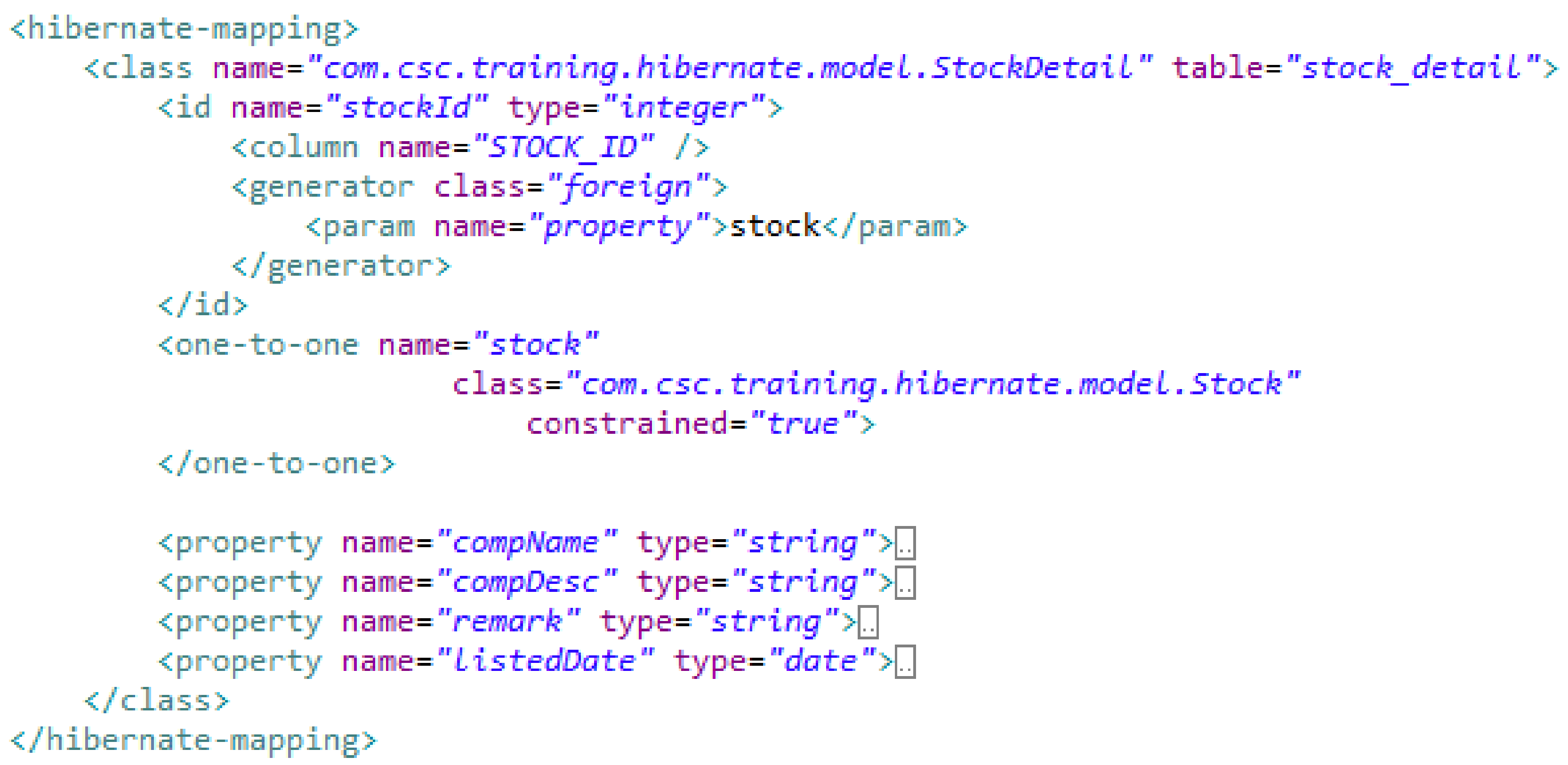


```java
public class StockDetail implements java.io.Serializable {

    private Integer stockId;
    private Stock stock;
    private String compName;
    private String compDesc;
    private String remark;
    private Date listedDate;

    public StockDetail() {
    }
```

Stock.hbm.xml

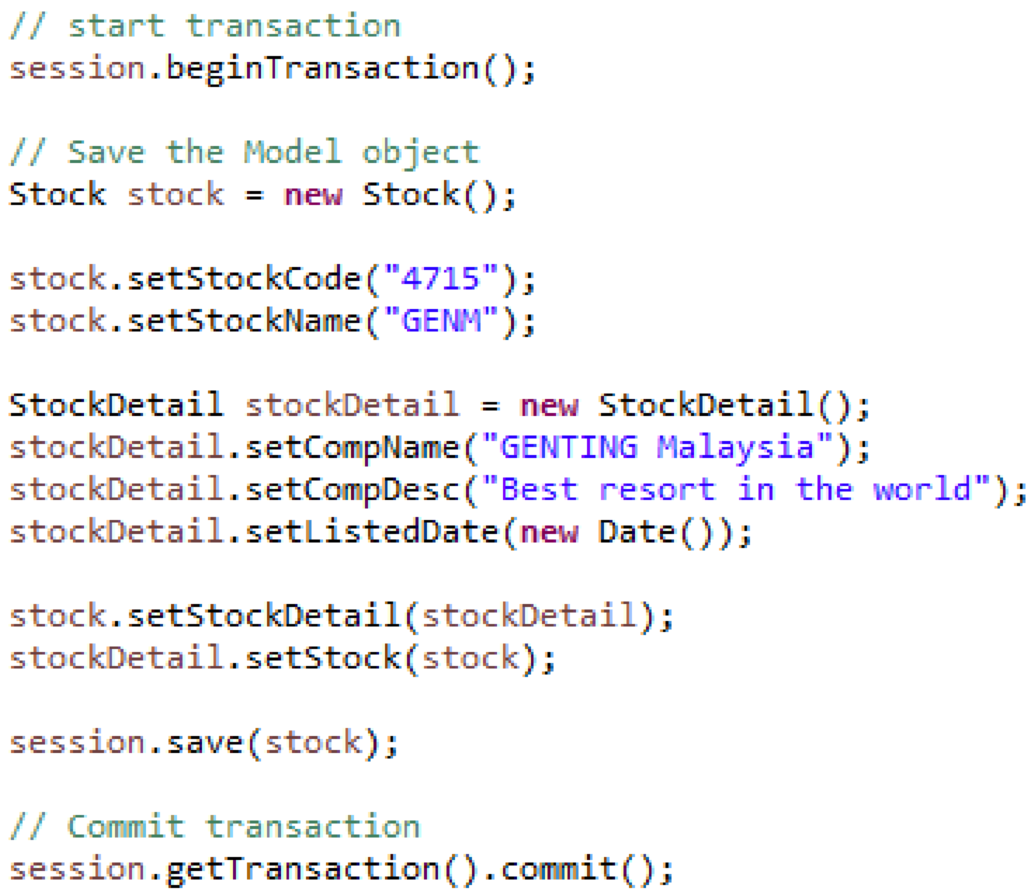
```
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.Stock" table="stock" >
        <id name="stockId" type="integer">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
            <column name="STOCK_CODE" length="10" not-null="true" unique="true" />
        </property>
        <property name="stockName" type="string">
            <column name="STOCK_NAME" length="20" not-null="true" unique="true" />
        </property>
        <one-to-one name="stockDetail"
                        class="com.csc.training.hibernate.model.StockDetail"
                            cascade="save-update">
        </one-to-one>
    </class>
</hibernate-mapping>
```

StockDetail.hbm.xml



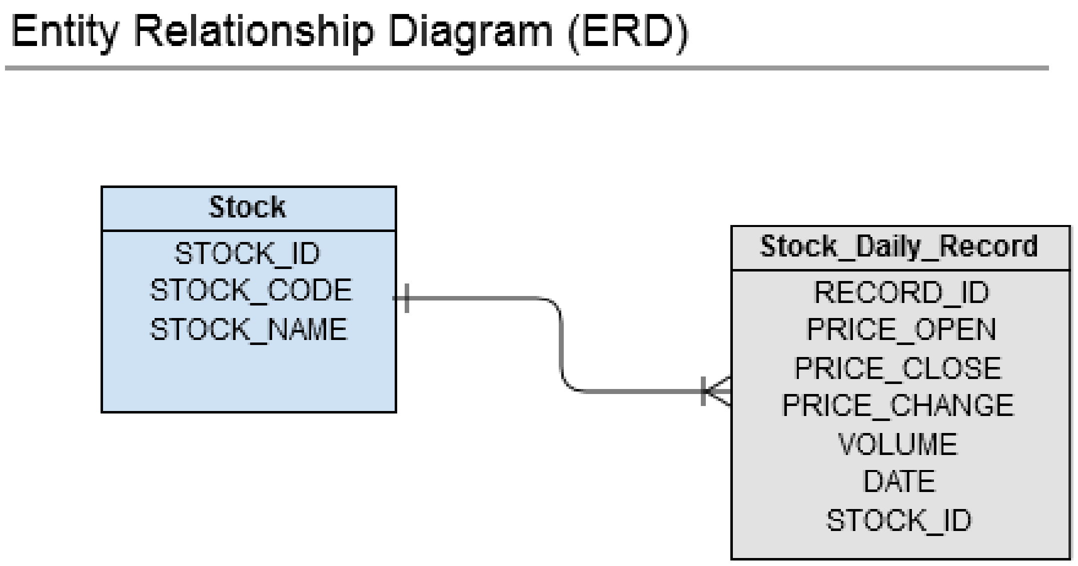
```
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.StockDetail" table="stock_detail">
        <id name="stockId" type="integer">
            <column name="STOCK_ID" />
            <generator class="foreign">
                <param name="property">stock</param>
            </generator>
        </id>
        <one-to-one name="stock"
                        class="com.csc.training.hibernate.model.Stock"
                            constrained="true">
        </one-to-one>

        <property name="compName" type="string">..
        <property name="compDesc" type="string">..
        <property name="remark" type="string">..
        <property name="listedDate" type="date">..
    </class>
</hibernate-mapping>
```

MainApp

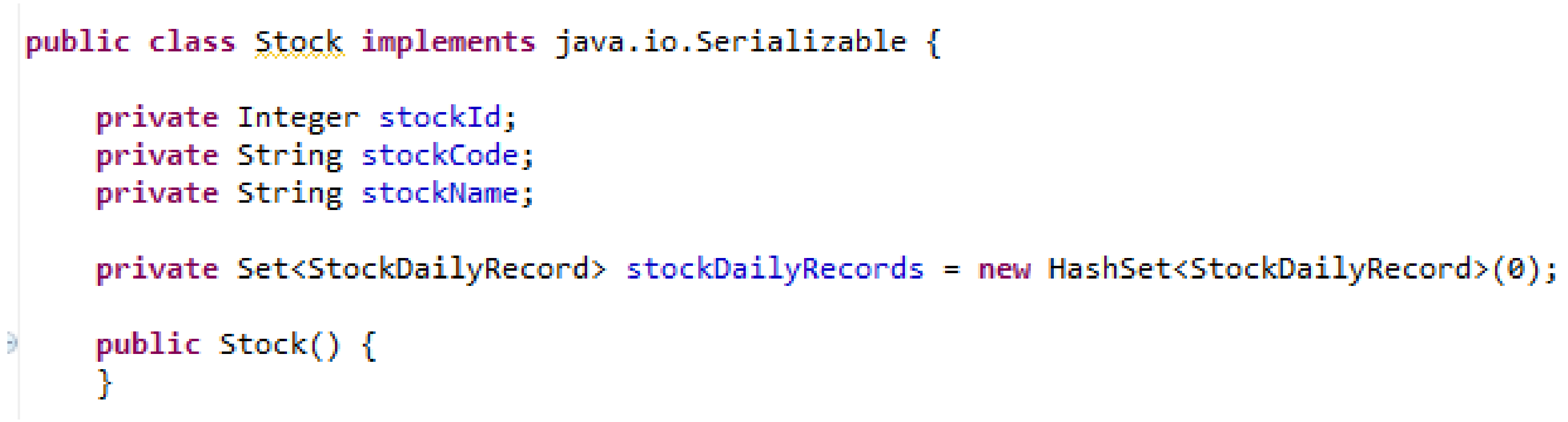


```
// start transaction
session.beginTransaction();

// Save the Model object
Stock stock = new Stock();

stock.setStockCode("4715");
stock.setStockName("GENM");

StockDetail stockDetail = new StockDetail();
stockDetail.setCompName("GENTING Malaysia");
stockDetail.setCompDesc("Best resort in the world");
stockDetail.setListedDate(new Date());

stock.setStockDetail(stockDetail);
stockDetail.setStock(stock);

session.save(stock);

// Commit transaction
session.getTransaction().commit();
```

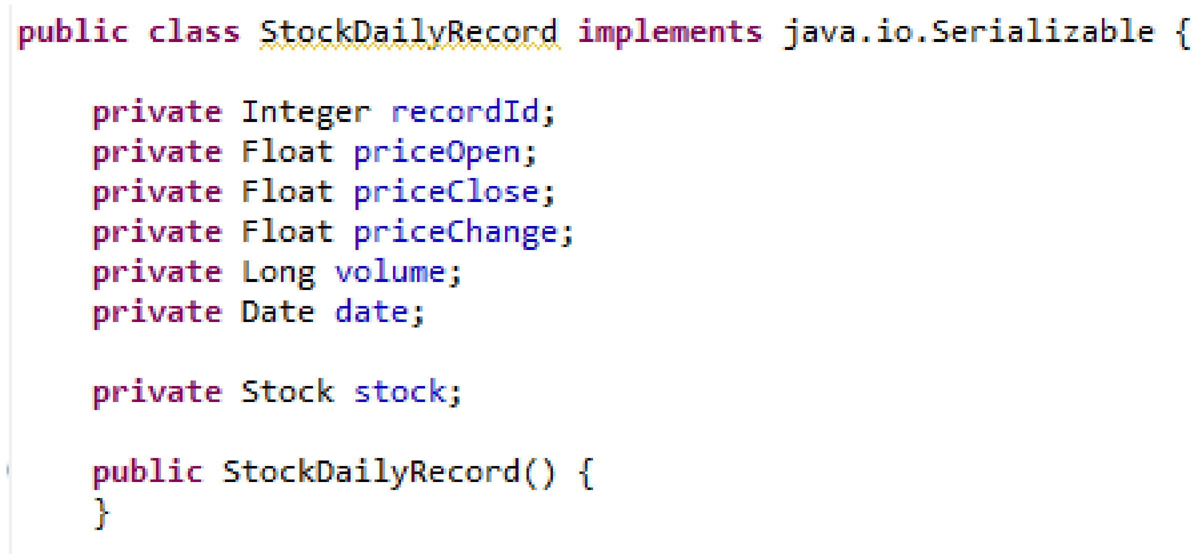
One-to-Many example (XML Mapping)



One-to-Many example (XML Mapping)

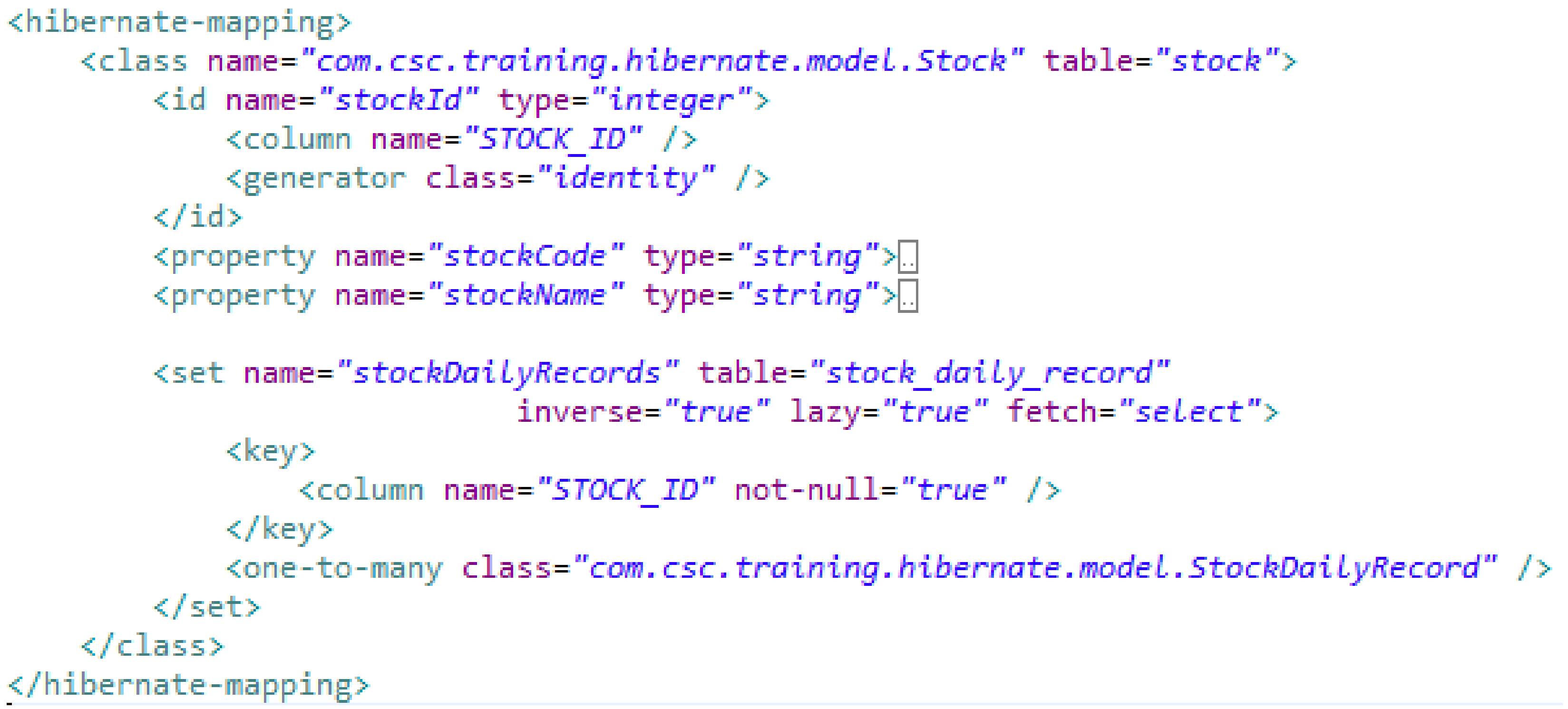


```
public class Stock implements java.io.Serializable {

    private Integer stockId;
    private String stockCode;
    private String stockName;

    private Set<StockDailyRecord> stockDailyRecords = new HashSet<StockDailyRecord>(0);

    public Stock() {
    }
```



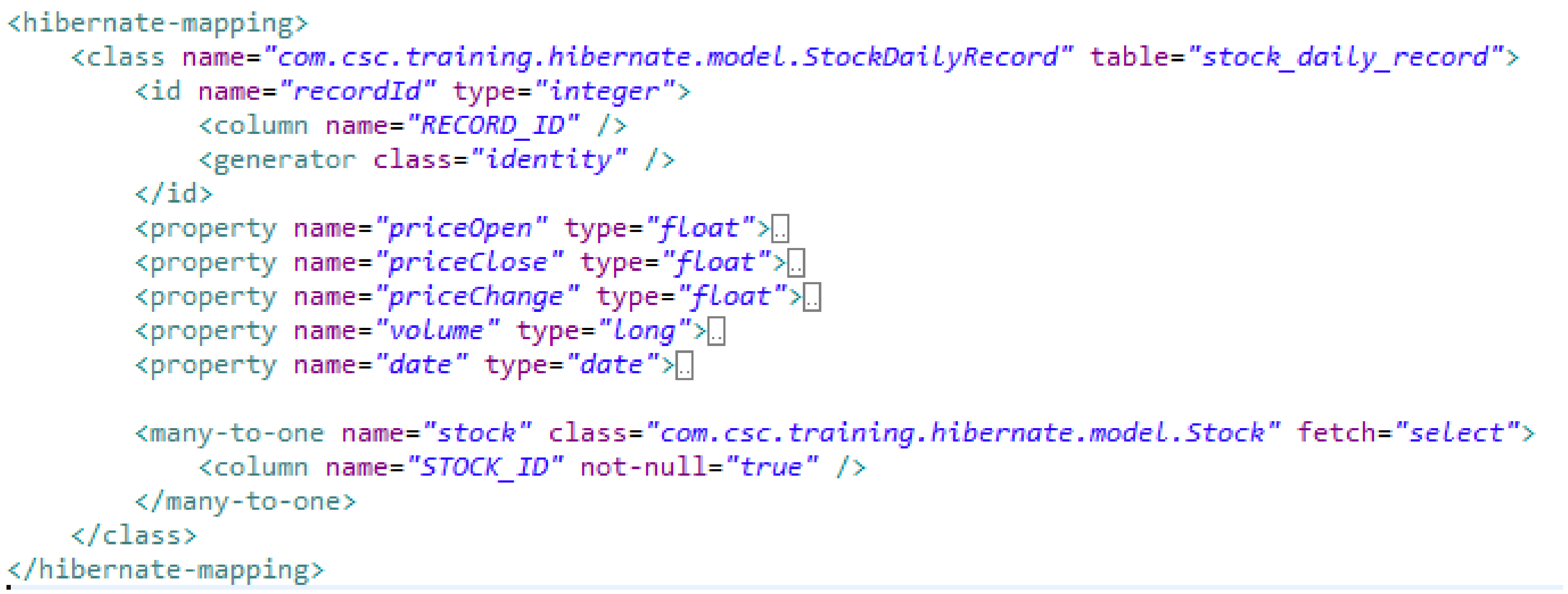
```
public class StockDailyRecord implements java.io.Serializable {

    private Integer recordId;
    private Float priceOpen;
    private Float priceClose;
    private Float priceChange;
    private Long volume;
    private Date date;

    private Stock stock;

    public StockDailyRecord() {
    }
```

Stock.hbm.xml



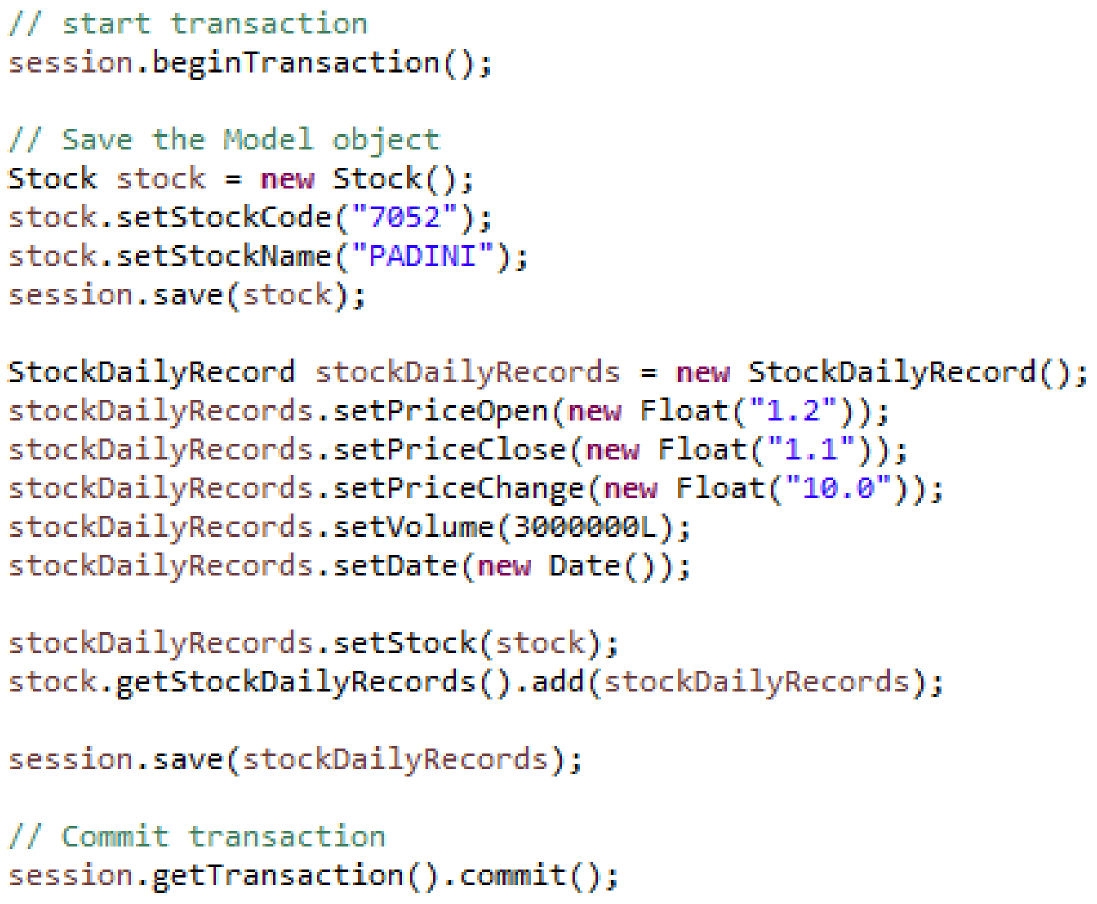
```
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.Stock" table="stock">
        <id name="stockId" type="integer">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">..
        <property name="stockName" type="string">..

        <set name="stockDailyRecords" table="stock_daily_record"
                            inverse="true" lazy="true" fetch="select">
            <key>
                <column name="STOCK_ID" not-null="true" />
            </key>
            <one-to-many class="com.csc.training.hibernate.model.StockDailyRecord" />
        </set>
    </class>
</hibernate-mapping>
```

StockDailyRecord.hbm.xml



```xml
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.StockDailyRecord" table="stock_daily_record">
        <id name="recordId" type="integer">
            <column name="RECORD_ID" />
            <generator class="identity" />
        </id>
        <property name="priceOpen" type="float">..
        <property name="priceClose" type="float">..
        <property name="priceChange" type="float">..
        <property name="volume" type="long">..
        <property name="date" type="date">..

        <many-to-one name="stock" class="com.csc.training.hibernate.model.Stock" fetch="select">
            <column name="STOCK_ID" not-null="true" />
        </many-to-one>
    </class>
</hibernate-mapping>
```

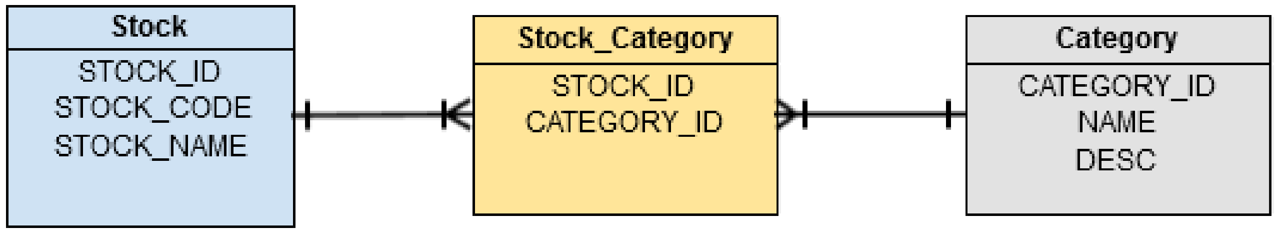
MainApp



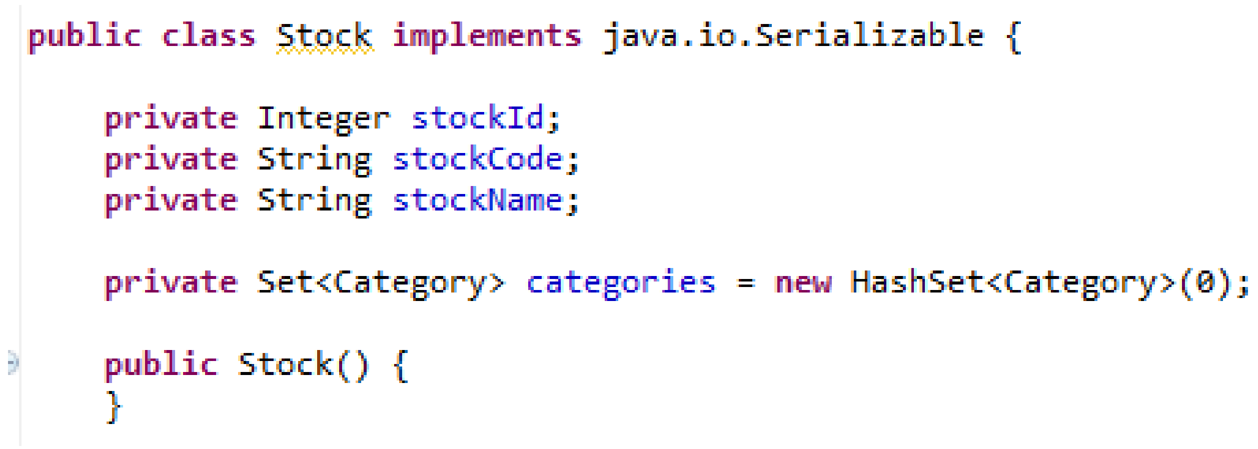
```
// start transaction
session.beginTransaction();

// Save the Model object
Stock stock = new Stock();
stock.setStockCode("7052");
stock.setStockName("PADINI");
session.save(stock);

StockDailyRecord stockDailyRecords = new StockDailyRecord();
stockDailyRecords.setPriceOpen(new Float("1.2"));
stockDailyRecords.setPriceClose(new Float("1.1"));
stockDailyRecords.setPriceChange(new Float("10.0"));
stockDailyRecords.setVolume(3000000L);
stockDailyRecords.setDate(new Date());

stockDailyRecords.setStock(stock);
stock.getStockDailyRecords().add(stockDailyRecords);

session.save(stockDailyRecords);

// Commit transaction
session.getTransaction().commit();
```

Many-to-Many example (XML Mapping)

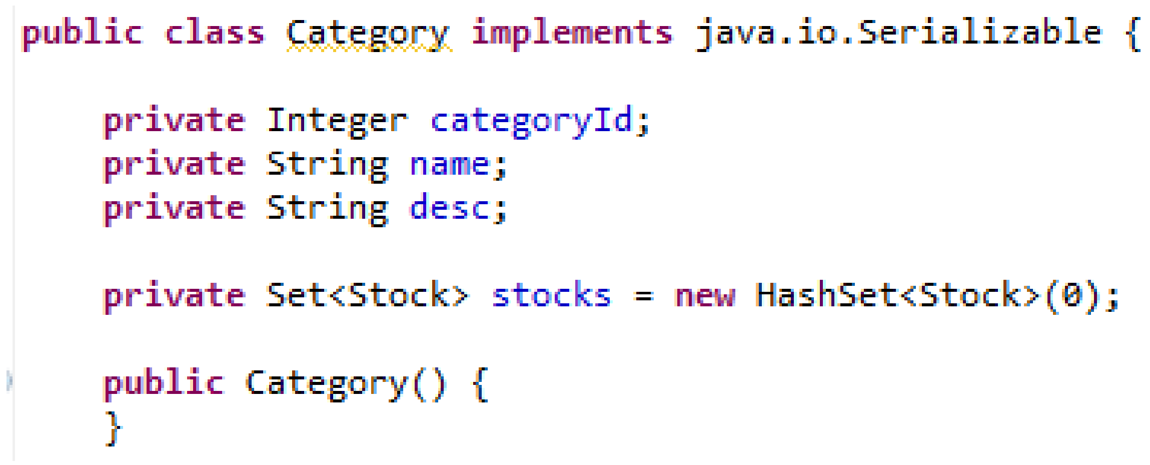
Entity Relationship Diagram (ERD)



Many-to-Many example (XML Mapping)

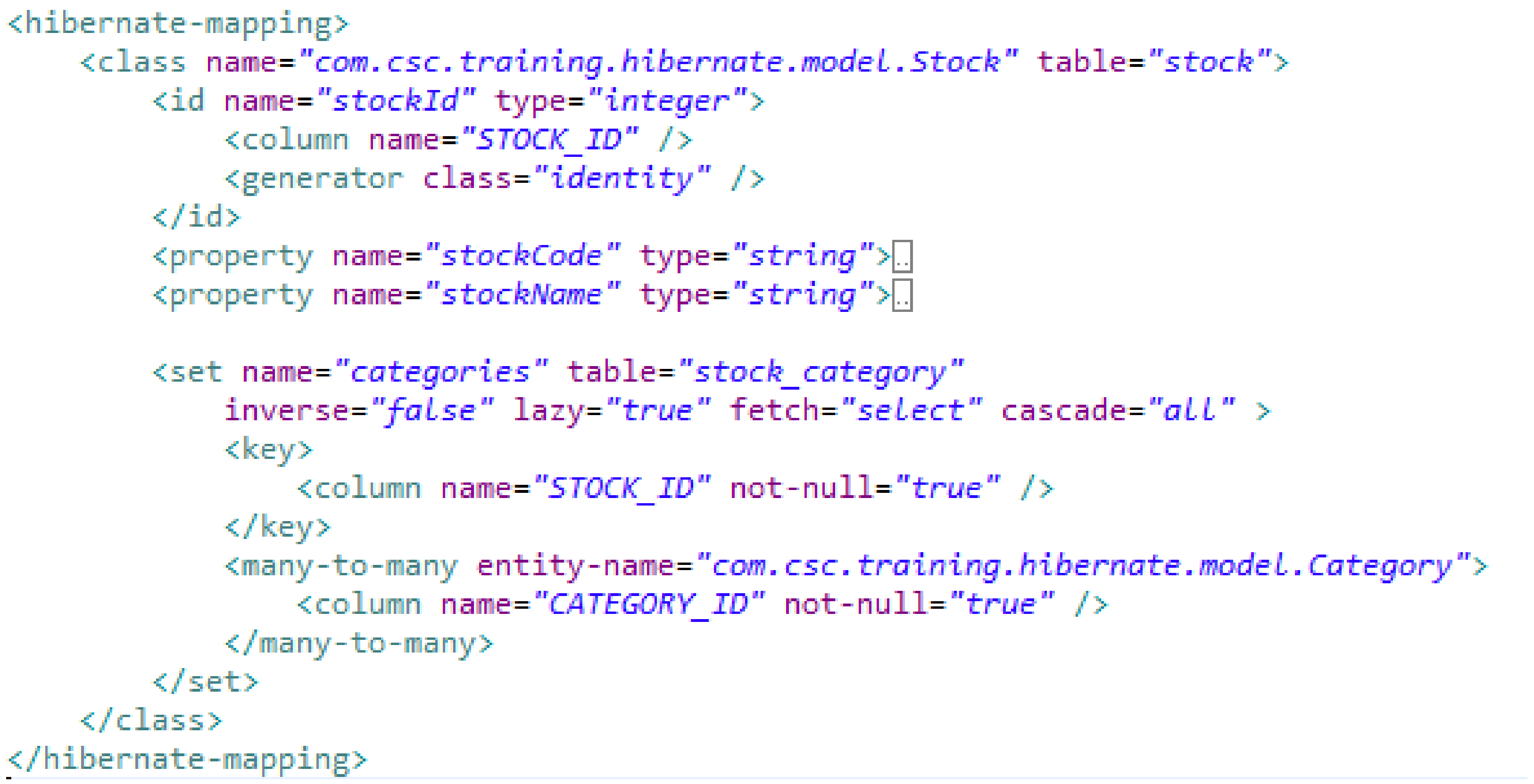


```java
public class Stock implements java.io.Serializable {

    private Integer stockId;
    private String stockCode;
    private String stockName;

    private Set<Category> categories = new HashSet<Category>(0);

    public Stock() {
    }
```



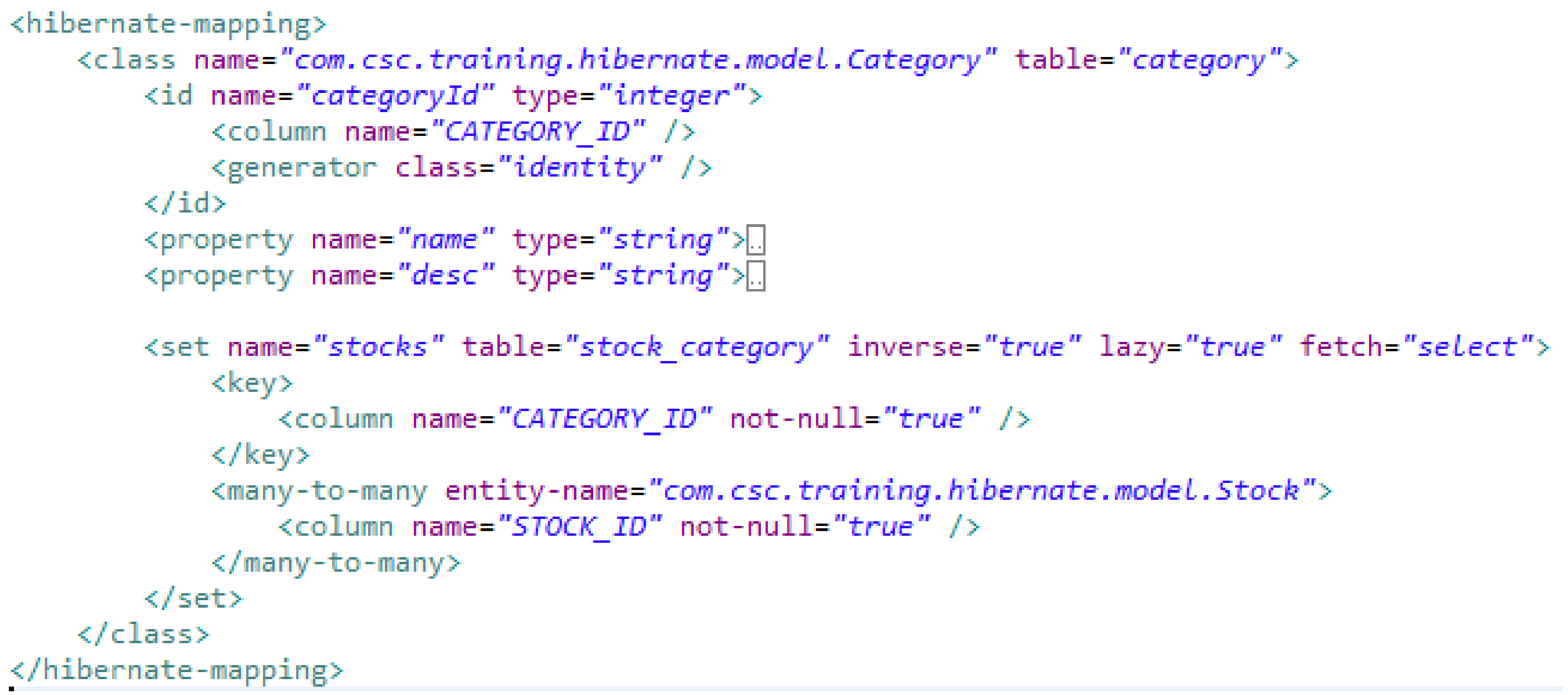
```java
public class Category implements java.io.Serializable {

    private Integer categoryId;
    private String name;
    private String desc;

    private Set<Stock> stocks = new HashSet<Stock>(0);

    public Category() {
    }
```

Stock.hbm.xml



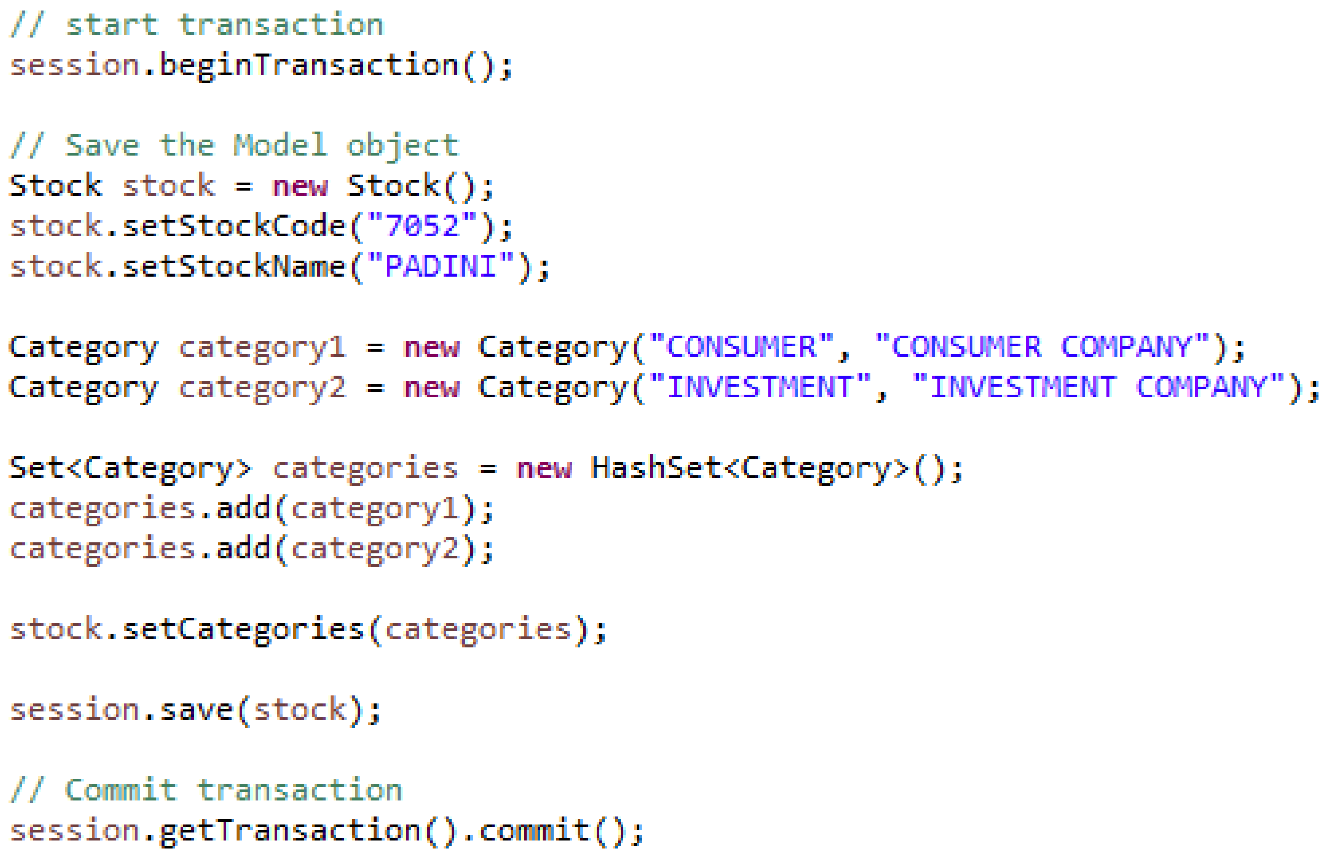
```xml
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.Stock" table="stock">
        <id name="stockId" type="integer">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
        <property name="stockName" type="string">

        <set name="categories" table="stock_category"
            inverse="false" lazy="true" fetch="select" cascade="all" >
            <key>
                <column name="STOCK_ID" not-null="true" />
            </key>
            <many-to-many entity-name="com.csc.training.hibernate.model.Category">
                <column name="CATEGORY_ID" not-null="true" />
            </many-to-many>
        </set>
    </class>
</hibernate-mapping>
```

Category.hbm.xml



```xml
<hibernate-mapping>
    <class name="com.csc.training.hibernate.model.Category" table="category">
        <id name="categoryId" type="integer">
            <column name="CATEGORY_ID" />
            <generator class="identity" />
        </id>
        <property name="name" type="string">
        <property name="desc" type="string">

        <set name="stocks" table="stock_category" inverse="true" lazy="true" fetch="select">
            <key>
                <column name="CATEGORY_ID" not-null="true" />
            </key>
            <many-to-many entity-name="com.csc.training.hibernate.model.Stock">
                <column name="STOCK_ID" not-null="true" />
            </many-to-many>
        </set>
    </class>
</hibernate-mapping>
```

MainApp



```java
// start transaction
session.beginTransaction();

// Save the Model object
Stock stock = new Stock();
stock.setStockCode("7052");
stock.setStockName("PADINI");

Category category1 = new Category("CONSUMER", "CONSUMER COMPANY");
Category category2 = new Category("INVESTMENT", "INVESTMENT COMPANY");

Set<Category> categories = new HashSet<Category>();
categories.add(category1);
categories.add(category2);

stock.setCategories(categories);

session.save(stock);

// Commit transaction
session.getTransaction().commit();
```

Spring Data JDBC

Problems of JDBC API

- Need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
- Need to perform exception handling code on the database logic
- Need to handle transaction
- Repetition of all these codes from one to another database logic is a time consuming task

What is Spring JDBC?

- Spring provides a simplification in handling database access with the Spring JDBC Template.
- The Spring JDBC template allows to clean-up the resources automatically, e.g. release the database connections.

Choosing an approach for JDBC database access

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcInsert and SimpleJdbcCall
- RDBMS Objects including MappingSqlQuery, SqlUpdate and StoredProcedure

JdbcTemplate

- The classic Spring JDBC approach and the most popular
- Performs some tasks
  - Executes SQL queries, update statements and stored procedure calls
  - Performs iteration over ResultSets and extraction of returned parameter values

JdbcTemplate

- Querying (SELECT)

```
int rowCount = this.jdbcTemplate.queryForObject("select count(*) from t_actor", Integer.class);
```

```
int countOfActorsNamedJoe = this.jdbcTemplate.queryForObject(
        "select count(*) from t_actor where first_name = ?", Integer.class, "Joe");
```

```
String lastName = this.jdbcTemplate.queryForObject(
        "select last_name from t_actor where id = ?",
        new Object[]{1212L}, String.class);
```

JdbcTemplate

- RowMapper
  - Map the ResultSet data to bean object in queryForObject() method

```java
public List<Actor> findAllActors() {
    return this.jdbcTemplate.query( "select first_name, last_name from t_actor", new ActorMapper());
}

private static final class ActorMapper implements RowMapper<Actor> {

    public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
        Actor actor = new Actor();
        actor.setFirstName(rs.getString("first_name"));
        actor.setLastName(rs.getString("last_name"));
        return actor;
    }
}
```

JdbcTemplate

- Updating (INSERT/UPDATE/DELETE) with jdbcTemplate

```
this.jdbcTemplate.update(
        "insert into t_actor (first_name, last_name) values (?, ?)",
        "Leonor", "Watling");
```

```
this.jdbcTemplate.update(
        "update t_actor set last_name = ? where id = ?",
        "Banjo", 5276L);
```

```
this.jdbcTemplate.update(
        "delete from actor where id = ?",
        Long.valueOf(actorId));
```

JdbcTemplate

- Other jdbcTemplate operations

```
this.jdbcTemplate.execute("create table mytable (id integer, name varchar(100))");
```

```
this.jdbcTemplate.update(
        "call SUPPORT.REFRESH_ACTORS_SUMMARY(?)",
        Long.valueOf(unionId));
```

NamedParameterJdbcTemplate

- Support for programming JDBC statements using named parameters

```
public int countOfActorsByFirstName(String firstName) {

    String sql = "select count(*) from T_ACTOR where first_name = :first_name";

    SqlParameterSource namedParameters = new MapSqlParameterSource("first_name", firstName);

    return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters, Integer.class);
}
```

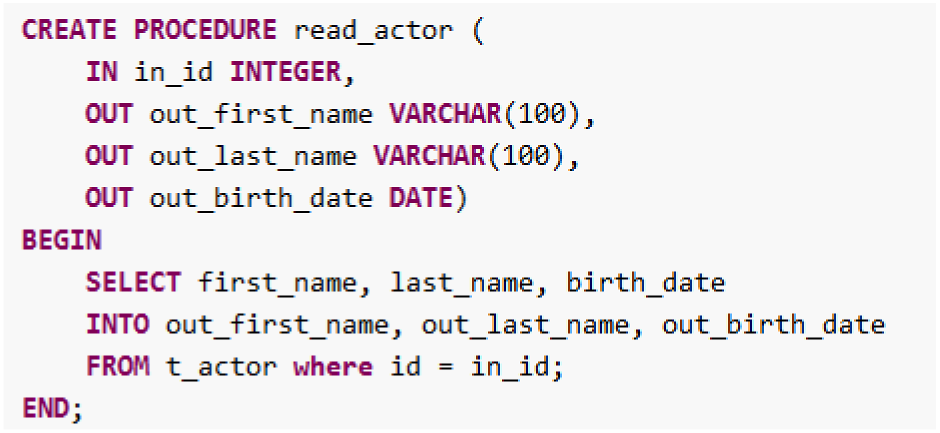
SimpleJdbcInsert

- Provides a simplified configuration by taking advantage of database metadata that can be retrieved through the JDBC driver

```java
public class JdbcActorDao implements ActorDao {

    private JdbcTemplate jdbcTemplate;
    private SimpleJdbcInsert insertActor;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.insertActor = new SimpleJdbcInsert(dataSource).withTableName("t_actor");
    }

    public void add(Actor actor) {
        Map<String, Object> parameters = new HashMap<String, Object>(3);
        parameters.put("id", actor.getId());
        parameters.put("first_name", actor.getFirstName());
        parameters.put("last_name", actor.getLastName());
        insertActor.execute(parameters);
    }

    // ... additional methods
}
```

SimpleJdbcCall

- Calling a stored procedure with SimpleJdbcCall



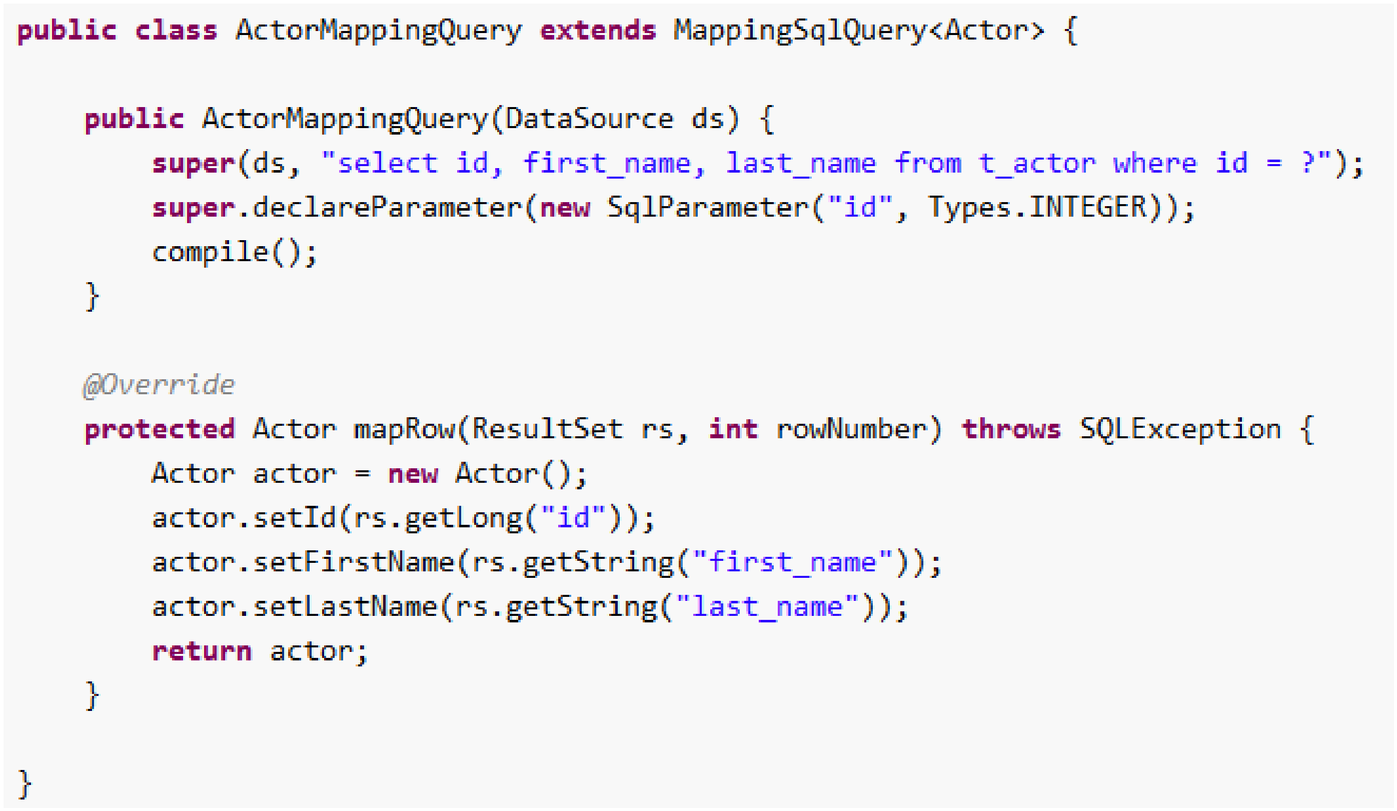
```
CREATE PROCEDURE read_actor (
    IN in_id INTEGER,
    OUT out_first_name VARCHAR(100),
    OUT out_last_name VARCHAR(100),
    OUT out_birth_date DATE)
BEGIN
    SELECT first_name, last_name, birth_date
    INTO out_first_name, out_last_name, out_birth_date
    FROM t_actor where id = in_id;
END;
```

SimpleJdbcCall

```
public class JdbcActorDao implements ActorDao {

    private JdbcTemplate jdbcTemplate;
    private SimpleJdbcCall procReadActor;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.procReadActor = new SimpleJdbcCall(dataSource)
                .withProcedureName("read_actor");
    }

    public Actor readActor(Long id) {
        SqlParameterSource in = new MapSqlParameterSource()
                .addValue("in_id", id);
        Map out = procReadActor.execute(in);
        Actor actor = new Actor();
        actor.setId(id);
        actor.setFirstName((String) out.get("out_first_name"));
        actor.setLastName((String) out.get("out_last_name"));
        actor.setBirthDate((Date) out.get("out_birth_date"));
        return actor;
    }
```

MappingSqlQuery

- MappingSqlQuery is a reusable query in which concrete subclasses must implement the abstract mapRow(..) method to convert each row of the supplied ResultSet into an object of the type specified.



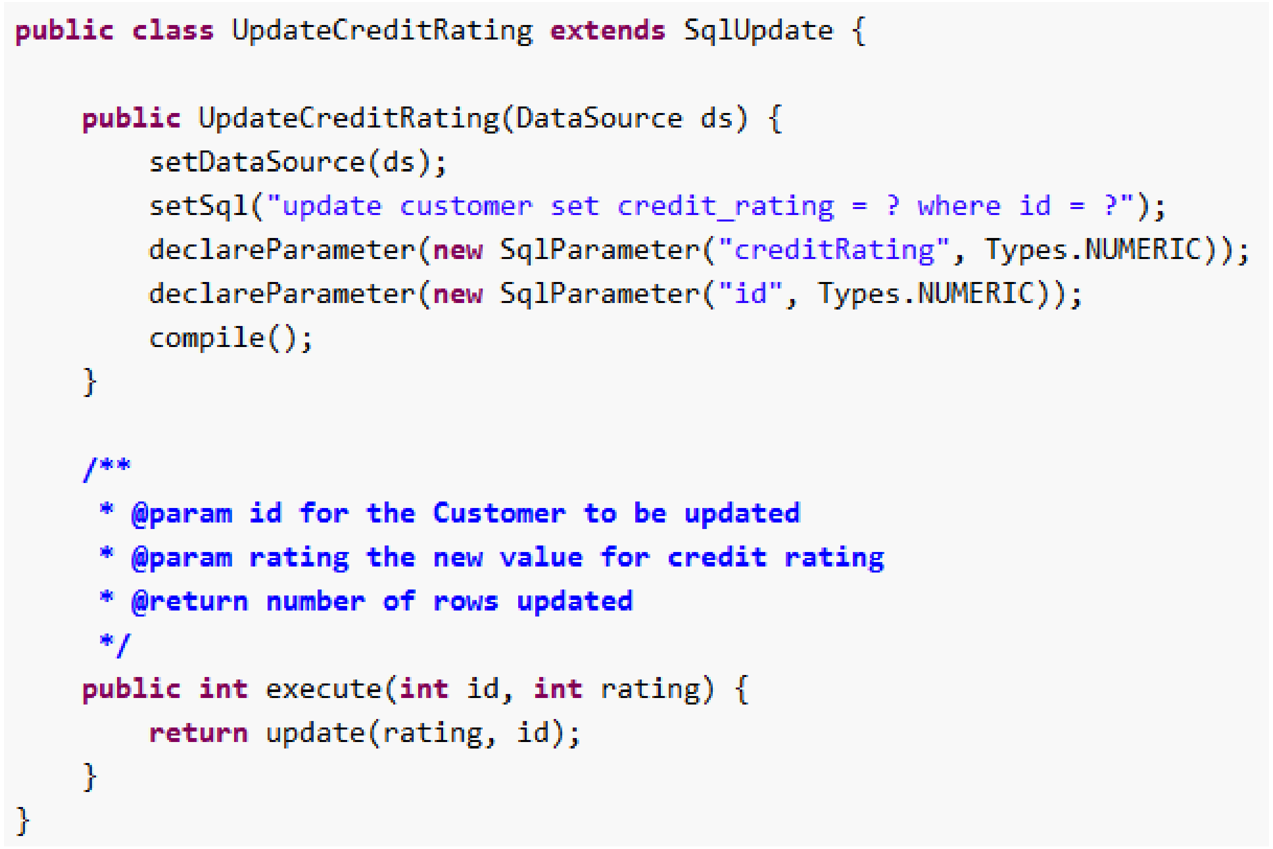
```
public class ActorMappingQuery extends MappingSqlQuery<Actor> {

    public ActorMappingQuery(DataSource ds) {
        super(ds, "select id, first_name, last_name from t_actor where id = ?");
        super.declareParameter(new SqlParameter("id", Types.INTEGER));
        compile();
    }

    @Override
    protected Actor mapRow(ResultSet rs, int rowNumber) throws SQLException {
        Actor actor = new Actor();
        actor.setId(rs.getLong("id"));
        actor.setFirstName(rs.getString("first_name"));
        actor.setLastName(rs.getString("last_name"));
        return actor;
    }

}
```

MappingSqlQuery

```java
private ActorMappingQuery actorMappingQuery;

@Autowired
public void setDataSource(DataSource dataSource) {
    this.actorMappingQuery = new ActorMappingQuery(dataSource);
}

public Customer getCustomer(Long id) {
    return actorMappingQuery.findObject(id);
}
```

SqlUpdate

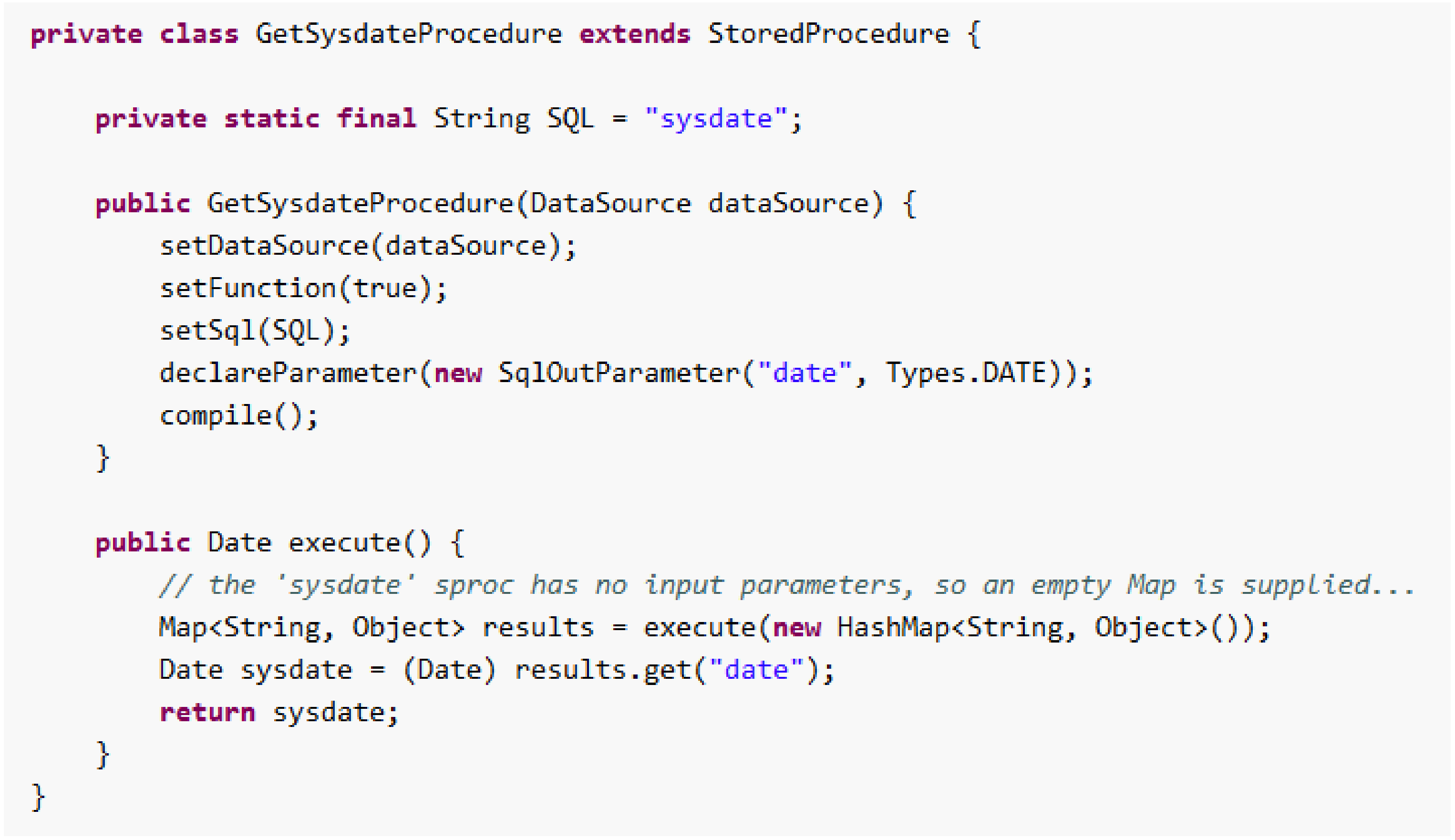
- The SqlUpdate class encapsulates an SQL update.
- Like a query, an update object is reusable



```java
public class UpdateCreditRating extends SqlUpdate {

    public UpdateCreditRating(DataSource ds) {
        setDataSource(ds);
        setSql("update customer set credit_rating = ? where id = ?");
        declareParameter(new SqlParameter("creditRating", Types.NUMERIC));
        declareParameter(new SqlParameter("id", Types.NUMERIC));
        compile();
    }

    /**
     * @param id for the Customer to be updated
     * @param rating the new value for credit rating
     * @return number of rows updated
     */
    public int execute(int id, int rating) {
        return update(rating, id);
    }
}
```

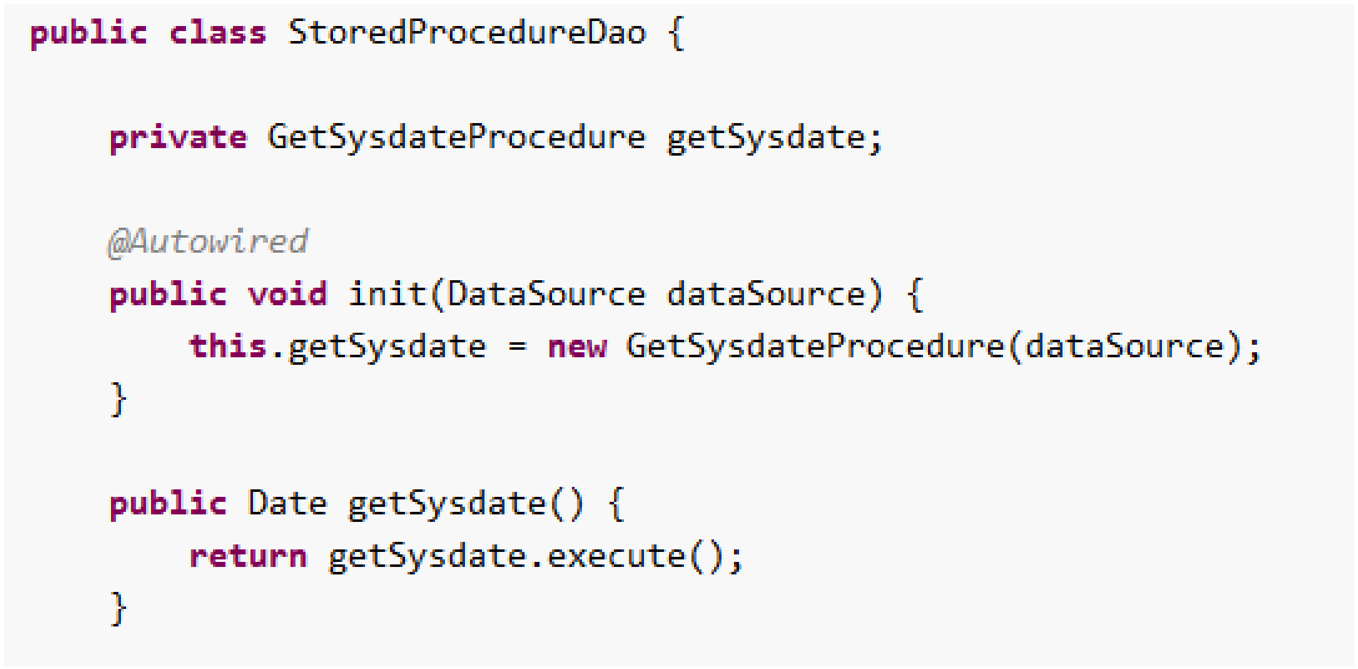
StoredProcedure

- The StoredProcedure class is a superclass for object abstractions of RDBMS stored procedures



```java
private class GetSysdateProcedure extends StoredProcedure {

    private static final String SQL = "sysdate";

    public GetSysdateProcedure(DataSource dataSource) {
        setDataSource(dataSource);
        setFunction(true);
        setSql(SQL);
        declareParameter(new SqlOutParameter("date", Types.DATE));
        compile();
    }

    public Date execute() {
        // the 'sysdate' sproc has no input parameters, so an empty Map is supplied...
        Map<String, Object> results = execute(new HashMap<String, Object>());
        Date sysdate = (Date) results.get("date");
        return sysdate;
    }
}
```

StoredProcedure



```java
public class StoredProcedureDao {

    private GetSysdateProcedure getSysdate;

    @Autowired
    public void init(DataSource dataSource) {
        this.getSysdate = new GetSysdateProcedure(dataSource);
    }

    public Date getSysdate() {
        return getSysdate.execute();
    }
```

Q&A

Thank You

Client Logo

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
11/11/2015	1.0	Initial Document	Kien Tran	

CSC

BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING