Hibernate

Agenda

- **Hibernate framework – An Overview**
- **Hibernate association and collection mapping**
- **Hibernate object lifecycle**
- **Hibernate transaction management**
- **Hibernate querying**
- **Hibernate HQL**
- **Java Persistence API**

Session 1: Introduction to Hibernate

Objectives

- *Refresher in enterprise application architectures*
- *Traditional persistence*
- *Hibernate motivation*

Objectives

- *Refresher in enterprise application architectures*
- *Traditional persistence*
- *Hibernate motivation*

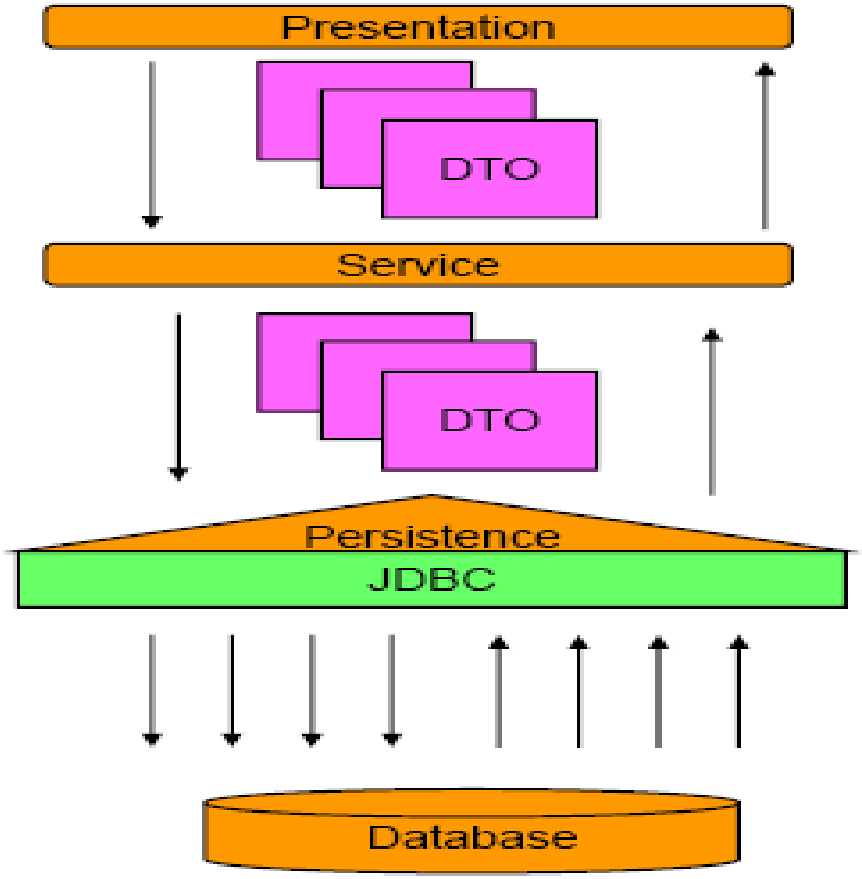
N-Tier Architecture

- Application is made up of layers or tiers
- Each layer encapsulates specific responsibilities
- Enables changes in one area with minimal impact to other areas of the application

N-Tier Architecture (Cont)

- **Common tiers**
  - Presentation
    - ‘View’ in model-view-controller
    - Responsible for displaying data only. No business logic
  - Service
    - Responsible for business logic
  - Persistence
    - Responsible for storing/retrieving data

N-Tier Architecture (Cont)



DAO Design Pattern

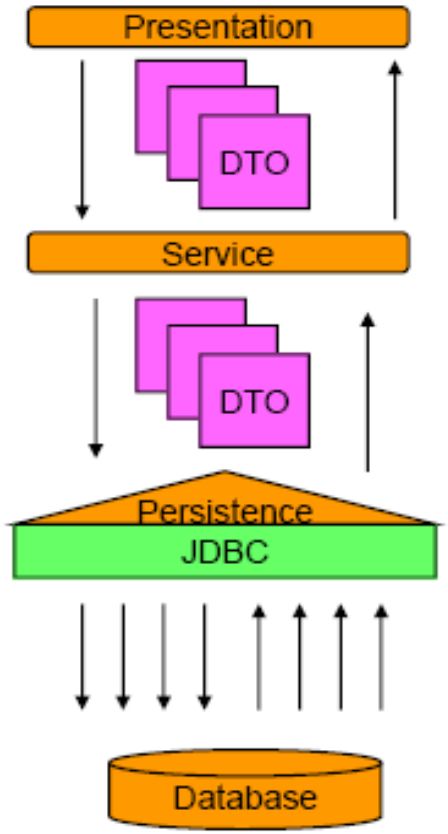
- Data Access Object
  - Abstracts CRUD (Create, Retrieve, Update, Delete) operations
- Benefits
  - Allows different storage implementations to be ‘plugged in’ with minimal impact to the rest of the system
  - Decouples persistence layer
  - Encourages and supports code reuse

Objectives

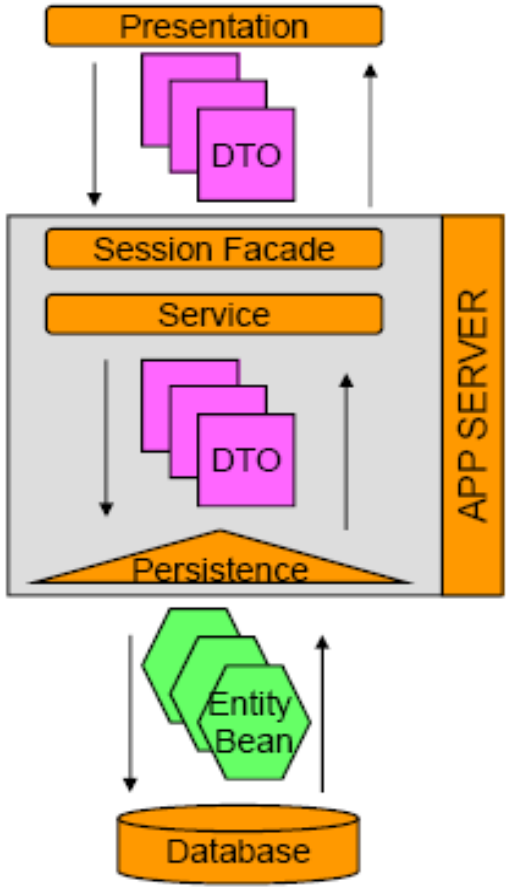
- *Refresher in enterprise application architectures*
- *Traditional persistence*
- *Hibernate motivation*

Traditional Persistence

Persistence with JDBC



Persistence with EJB 2.x



JDBC Overview

- JDBC API provides ability to
  - Establish connection to a database
  - Execute SQL statements
  - Create parameterized queries
  - Iterate through results
  - Manage database transactions

JDBC Overview (Cont)

- **Basic Steps to JDBC Operations**
  - Load driver or obtain datasource
  - Establish connection using a JDBC URL
  - Create statement
  - Execute statement
  - Optionally, process results in result set
  - Close database resources
  - Optionally, commit/rollback transaction

JDBC Example – Create Account

```
public Account createAccount(Account account) {
Connection connection = null;
PreparedStatement getAccountIdStatement = null;
PreparedStatement createAccountStatement = null;
ResultSet resultSet = null;
long accountId=0;
// Load driver
try {
Class.forName("oracle.jdbc.driver.OracleDriver");
catch (Exception e) {
throw new RuntimeException(e);
}
try {
//Get connection and set auto commit to false
Connection connection =
DriverManager.getConnection("jdbc:oracle:
thin:lecture1/lecture1@localhost:1521:XE");
connection.setAutoCommit(false);
...
```

JDBC Example – Create Account(Cont)

```
...
//Get account id from sequence
getAccountIdStatement = connection
.prepareStatement("SELECT ACCOUNT_ID_SEQ.NEXTVAL
FROM DUAL");
resultSet = getAccountIdStatement.executeQuery();
resultSet.next();
accountId = resultSet.getLong(1);
//Create the account
createAccountStatement = connection
.prepareStatement(AccountDAOConstants.CREATE_ACCOUNT);
createAccountStatement.setLong(1, accountId);
createAccountStatement.setString(2,
account.getAccountType());
createAccountStatement.setDouble(3, account.getBalance());
createAccountStatement.executeUpdate();
//Commit transaction
connection.commit();
}
...
```

JDBC Example – Create Account(Cont)

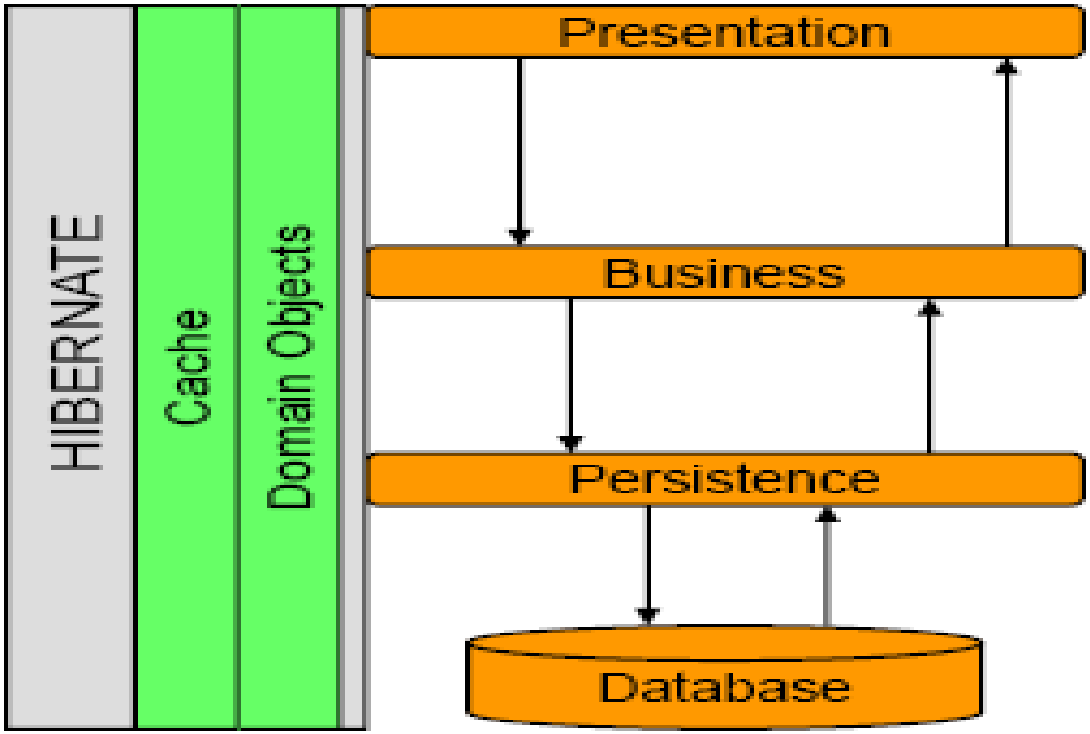
```
...
catch (SQLException e) {
//In case of exception, rollback
try{
connection.rollback();
}catch(SQLException e1){// log error}
throw new RuntimeException(e);
}
finally {
//close database resources
try {
if (resultSet != null)
resultSet.close();
if (getAccountIdStatement!= null)
getAccountIdStatement.close();
if (createAccountStatement!= null)
createAccountStatement.close();
if (connection != null)
connection.close();
} catch (SQLException e) {// log error}
}
}
```

Objectives

- *Refresher in enterprise application architectures*
- *Traditional persistence*
- *Hibernate motivation*

Traditional Persistence vs. Hibernate

Persistence with Hibernate



Hibernate History

- Grass roots development (2001)
  - Christian Bauer
  - Gavin King
- JBoss later hired lead Hibernate developers (2003)
  - Brought Hibernate under the Java EE specification
  - Later officially adopted as the official EJB3.0 persistence
- implementation for the JBoss application server.
- EJB 3.0 Expert Group (2004)
  - Key member which helped shape EJB3.0 and JPA
- NHibernate
  - .NET version release in 2005

Why Hibernate?

- Impedance mismatch
  - Object-oriented vs. relational
- Failure of EJB 2.x
  - Entity Beans were extremely slow, complex
- Reduce application code by 30%
  - Less code is better maintainable

Why Hibernate?

- Java developers are not database developers
  - Reduce the need for developers to know and fully understand database design, SQL, performance tuning
  - Increase portability across database vendors
- Increase performance by deferring to experts
  - Potential decrease in database calls
  - More efficient SQL statements
  - Hibernate cache usage

Summary

- Refresher in application architectures
- Traditional Persistent implementation
  - JDBC example
- Motivation
  - Origination and history of Hibernate
  - Reasons for Hibernates development
    - Impedance mismatch
    - Failure of EJB 2.x
    - Java developers are not database developers
    - Performance benefits

Session 2: Walk-through of a Simple Hibernate Example

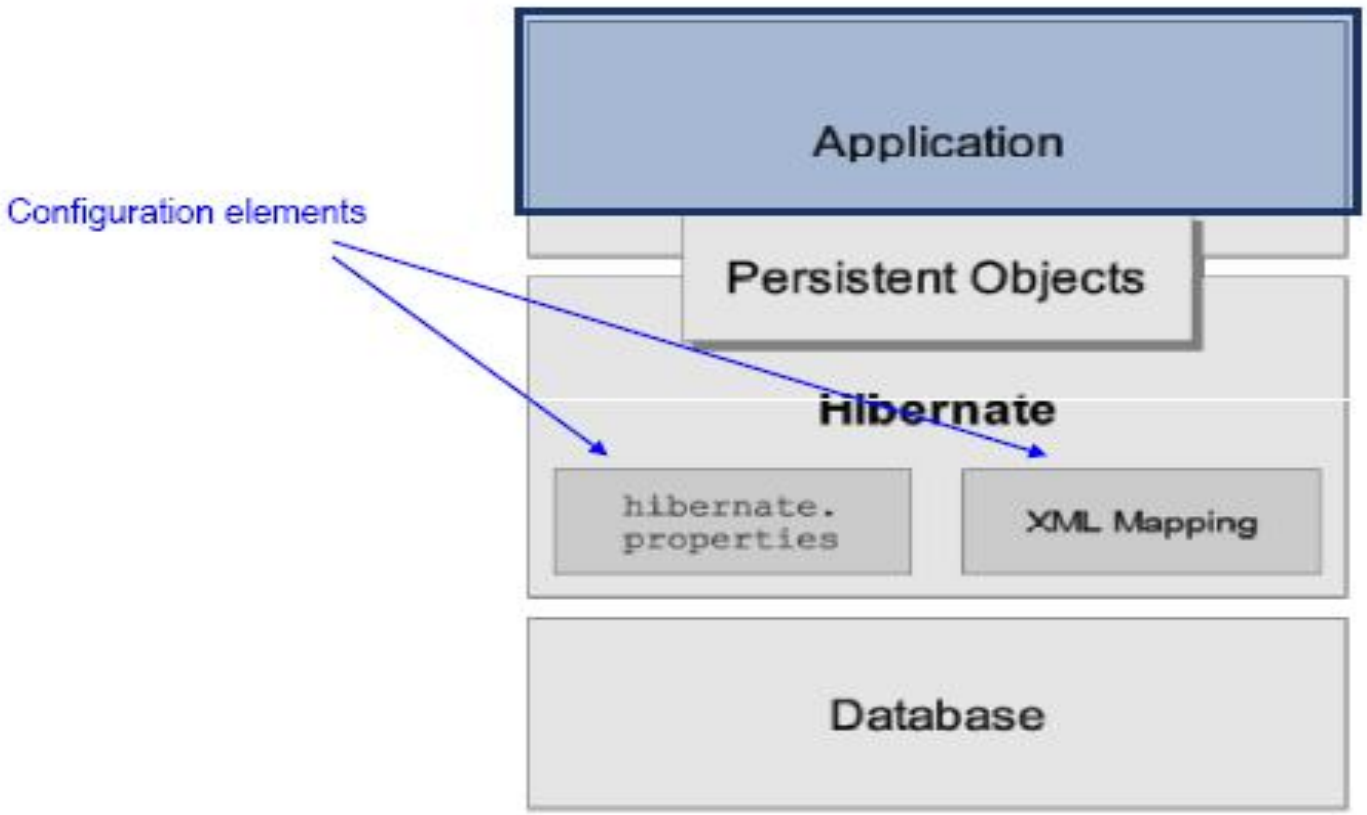
Objectives

- Creating a simple, but full, end to end Hibernate Application
- How to use JUnit in developing Enterprise Application

Objectives

- Creating a simple, but full, end to end Hibernate Application
- How to use JUnit in developing Enterprise Application

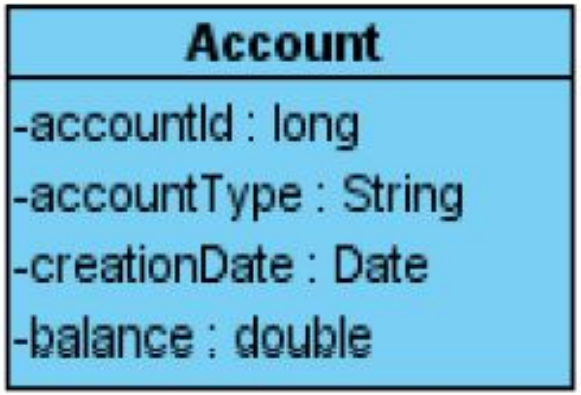
High Level Architecture



Building a Hibernate Application

1. Define the domain model
2. Setup your Hibernate configuration
   - hibernate.cfg.xml
3. Create the domain object mapping files
   - <domain_object>.hbm.xml
4. Make Hibernate aware of the mapping files
   - Update the hibernate.cfg.xml with list of mapping files
5. Implement a HibernateUtil class
   - Usually taken from the Hibernate documentation
6. Write your code

Account Object / Table



ACCOUNT TABLE

Column Name	Data Type	Nullable	Default	Primary Key
ACCOUNT_ID	NUMBER	No	-	1
ACCOUNT_TYPE	VARCHAR2(200)	No	-	-
CREATION_DATE	TIMESTAMP(6)	No	-	-
BALANCE	NUMBER	No	-	-

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
. . .
</session-factory>
<hibernate-configuration>
```

Configure Hibernate here, particularly the session-factory

hibernate.cfg.xml (Cont)

```
...
<session-factory>
<property name="hibernate.connection.driver_class">
oracle.jdbc.driver.OracleDriver</property>
<property
name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE
</property>
<property name="hibernate.connection.username">lecture2</property>
<property name="hibernate.connection.password">lecture2</property>
...
```

hibernate.cfg.xml (Cont)

```
...
<property name="dialect">
org.hibernate.dialect.Oracle10gDialect</property>
<property name="connection.pool_size">1</property>
<property name="current_session_context_class">thread</property>
<property name="show_sql">true</property>
<property name="format_sql">false</property>
</session-factory>

...
```

Configuring Hibernate

- There are multiple ways to configure Hibernate, and an application can leverage multiple methods at once

- Hibernate will look for and use configuration properties in the following order
  - hibernate.properties (when 'new Configuration()' is called)
  - hibernate.cfg.xml (when 'configure()' is called on Configuration)
  - Programatic Configuration Settings

Configuring Hibernate(Cont)

```
SessionFactory sessionFactory =
  new Configuration()
    .configure("hibernate.cfg.xml")
      .setProperty(Environment.DefaultSchema,"MY_SCHEMA");
```

Initialize w/ Hibernate.properties

Load XML properties, overriding previous

Programatically set 'Default Schema', overidding all previous settings for this value

Object Mapping Files

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class>
. . .
</class>
</hibernate-mapping>
```

Describe your class attributes here.

Account.hbm.xml Mapping File

```
...
<class name="courses.hibernate.vo.Account“ table="ACCOUNT">
<id name="accountId" column="ACCOUNT_ID">
        <generator class="native"/>
</id>
<property name="creationDate" column="CREATION_DATE“
type="timestamp" update="false"/>
<property name="accountType" column="ACCOUNT_TYPE“ type="string"
update="false"/>
<property name="balance" column="BALANCE“ type="double"/>
</class>
...
```

Hibernate ID Generators

- Native:
  - Leverages underlying database method for generating ID (sequence, identity, etc…)
- Increment:
  - Automatically reads max value of identity column and increments by 1
- UUID:
  - Universally unique identifier combining IP & Date (128-bit)
- Many more…

Make Hibernate aware of the mapping files

```
...
<property name="dialect">org.hibernate.dialect.Oracle10gDialect
</property>
<property name="connection.pool_size">1</property>
<property name="current_session_context_class">thread</property>
<property name="show_sql">true</property>
<property name="format_sql">false</property>
<mapping resource="Account.hbm.xml"/>
</session-factory>
...
```

HibernateUtil

- Convenience class to handle building and obtaining the Hibernate SessionFactory
  - Use recommended by the Hibernate org
- SessionFactory is thread-safe
  - Singleton for the entire application
- Used to build Hibernate ‘Sessions’
  - Hibernate Sessions are NOT thread safe
  - One per thread of execution

HibernateUtil (Cont)

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil  {
    private static final SessionFactory sessionFactory;
    // initialize sessionFactory singleton
    static {
    sessionFactory = new Configuration().
    configure().buildSessionFactory();
    }
    // method used to access singleton
    public static SessionFactory getSessionFactory() {
    return sessionFactory;
    }
}
```

Session API

- How to get Hibernate Session instance

```
Session session =HibernateUtil.getSessionFactory().getCurrentSession();
```

- Hibernate Session
  - session.saveOrUpdate()
  - session.get()
  - session.delete()
- What about just plain save?
  - It's there, but not typically used
  - session.save()

Account DAO – saveOrUpdate()

```
public void saveOrUpdateAccount(Account account) {
    Session session =
    HibernateUtil.getSessionFactory().getCurrentSession();
    session.saveOrUpdate(account);
}
```

Remember the number of LOC needed to do this with JDBC?

Account DAO – get()

```
public Account getAccount(long accountId) {
    Session session =
    HibernateUtil.getSessionFactory().getCurrentSession();
    Account account =
    (Account)session.get(Account.class,accountId);
    return account;
}
```

Account DAO – delete()

```
public void deleteAccount(Account account) {
    Session session =
    HibernateUtil.getSessionFactory().getCurrentSession();
    session.delete(account);
}
```

Service Layer - AccountService

```
import courses.hibernate.dao.AccountDAO;
import courses.hibernate.vo.Account;
/**
 * Service layer for Account
 */
public class AccountService {
        AccountDAO accountDAO = new AccountDAO();
        ...
}
```

Declare all business methods here

AccountService – business methods

```
/**
 * Create a new account or update an existing one
 * @param account
 *          account to be persisted
*/
public void saveOrUpdateAccount(Account account) {
            accountDAO.saveOrUpdateAccount(account);
}
```

AccountService – business methods

```
/**
 * Retrieve an account
 * @param accountId
 *          identifier of the account to be retrieved
 * @return account represented by the identifier provided
*/
public Account getAccount(long accountId) {
            return accountDAO.getAccount(accountId);
}
```

AccountService – business methods

```
/**
 * Delete account
 * @param account
 *          account to be deleted
*/
public void deleteAccount(Account account) {
            accountDAO.deleteAccount(account);
}
```

Testing with JUnit

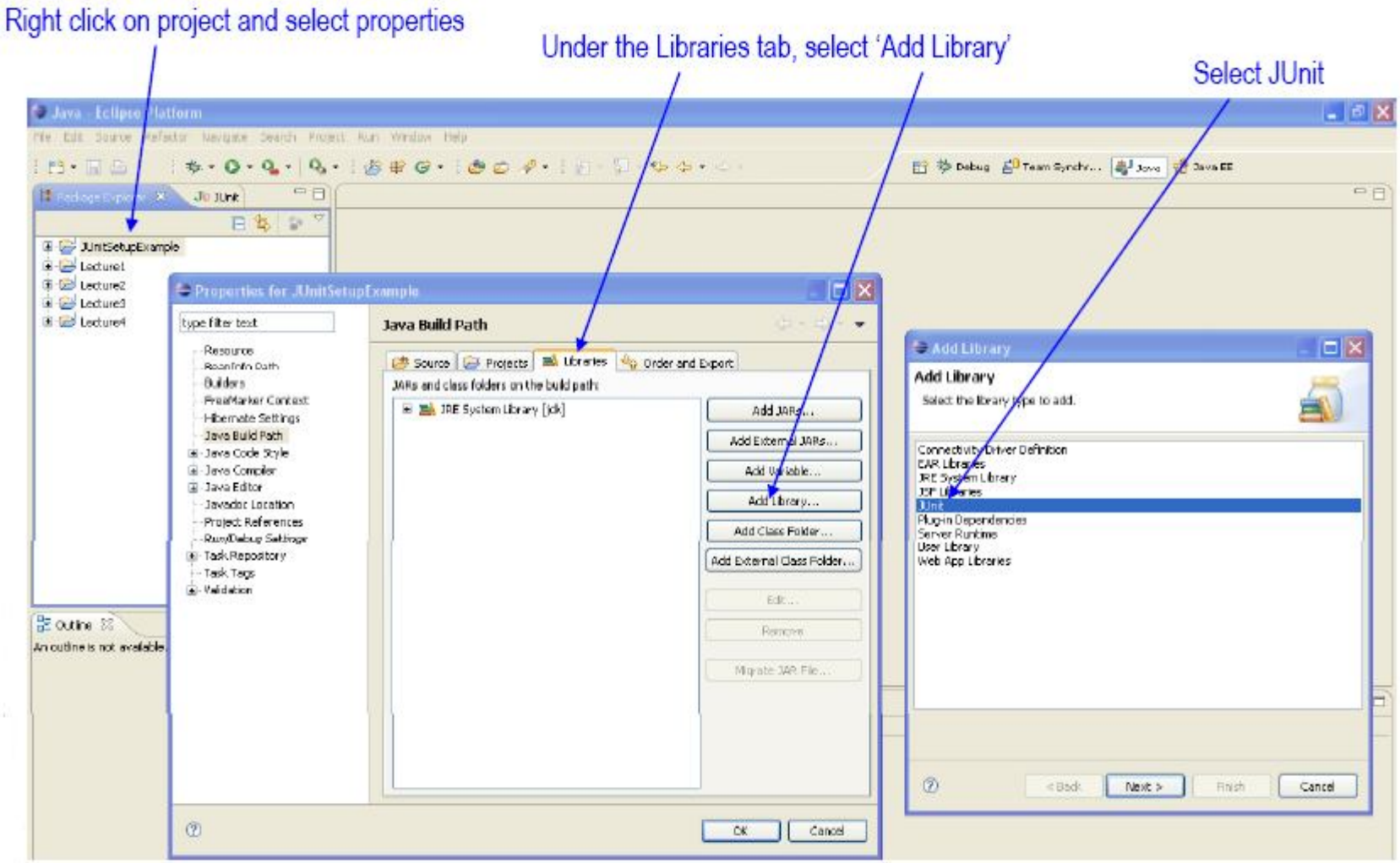
- JUnit is an open source framework to perform testing against units of code.
  - A single test class contains several test methods
  - Provides helper methods to make ‘assertions’ of expected results
  - Common to have multiple test classes for an application

Using JUnit

- Download the jar from JUnit.org
- Add downloaded jar to project classpath
- Create a class to house your test methods, naming it anything you like (typically identifying it as a test class)
- Implement test methods, naming them anything you like and marking each with the @Test annotation at the method level
- Call the code to be tested passing in known variables and based on expected behavior, use 'assert' helper methods provided by Junit to verify correctness
  - **Assert.assertTrue(account.getAccountId() == 0);**

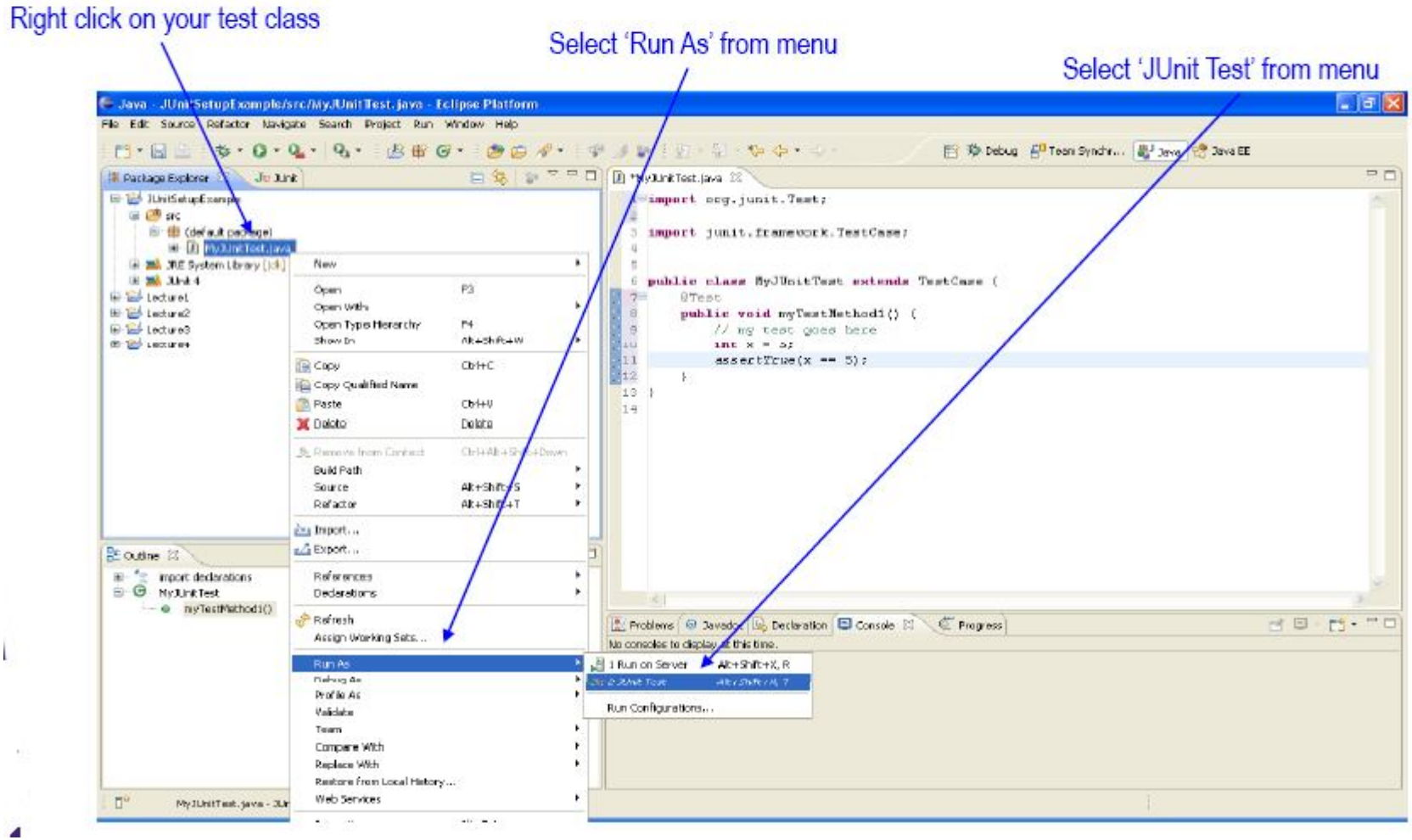
JUnit and Eclipse

- **JUnit comes with most Eclipse downloads**



JUnit and Eclipse

- **Running JUnit in Eclipse**



Test Create

```
@Test
public void testCreateAccount() {
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    Account account = new Account();
    // no need to set id, Hibernate will do it for us
    account.setAccountType(Account.ACCOUNT_TYPE_SAVINGS);
    account.setCreationDate(new Date());
    account.setBalance(1000L);
    // confirm that there is no accountId set
    Assert.assertTrue(account.getAccountId() == 0);
...
```

Test Create (Cont)

```
. . .
    // save the account
    AccountService accountService = new AccountService();
    accountService.saveOrUpdateAccount(account);
    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();
    System.out.println(account);
    // check that ID was set after the hbm session
    Assert.assertTrue(account.getAccountId() > 0);
}
```

Handling Transactions

- Why am I starting/ending my transactions in my test case?
  - In order to take advantage of certain Hibernate features, the Hibernate org recommends you close your transactions as late as possible. For test cases, this means in the tests themselves
  - Later we'll discuss suggested ways of handling this within applications

Test Get

```
@Test
public void testGetAccount() {
    Account account = createAccount(); // create account to get
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    AccountService accountService = new AccountService();
    Account anotherCopy =
    accountService.getAccount(account.getAccountId());
    // make sure these are two separate instances
    Assert.assertTrue(account != anotherCopy);
    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();
}
```

Test Update Balance

```
@Test
public void testUpdateAccountBalance() {
// create account to update
Account account = createAccount();
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
AccountService accountService = new AccountService();
account.setBalance(2000);
accountService.saveOrUpdateAccount(account);
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
...
```

Test Update Balance (Cont)

```
. . .
Session session2 = HibernateUtil.getSessionFactory().getCurrentSession();
session2.beginTransaction();
Account anotherCopy = accountService.getAccount(account.getAccountId());
System.out.println(anotherCopy);
// make sure the one we just pulled back from the
// database has the updated balance
Assert.assertTrue(anotherCopy.getBalance() == 2000);
session2.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
}
```

Test Delete

```
@Test
public void testDeleteAccount() {
// create an account to delete
Account account = createAccount();
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
AccountService accountService = new AccountService();
// delete the account
accountService.deleteAccount(account);
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
...
```

Test Delete (Cont)

```
...
Session session2 = HibernateUtil.getSessionFactory().getCurrentSession();
session2.beginTransaction();
// try to get the account again -- should be null
Account anotherCopy = accountService.getAccount(account.getAccountId());
System.out.println("var anotherCopy = "+ anotherCopy);
Assert.assertNull(anotherCopy);
session2.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
}
```

Summary

- End to end Hibernate Application
  - Configuration

    hibernate.cfg.xml

```
<property
name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE
</property>
```

  - Object mapping files

    Account.hbm.xml

```
<class name="courses.hibernate.vo.Account“
table="ACCOUNT">
<property name="accountType“ column="ACCOUNT_TYPE“ type="string"
update="false"/>
```

Summary (Cont)

– HibernateUtil to handle Session

```
static {
    sessionFactory = newConfiguration()
    .configure().buildSessionFactory();
}
public static SessionFactory getSessionFactory() {
    return sessionFactory;
}
```

– Writing the implementation

```
Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
session.saveOrUpdate(account);
```

Q & A