2

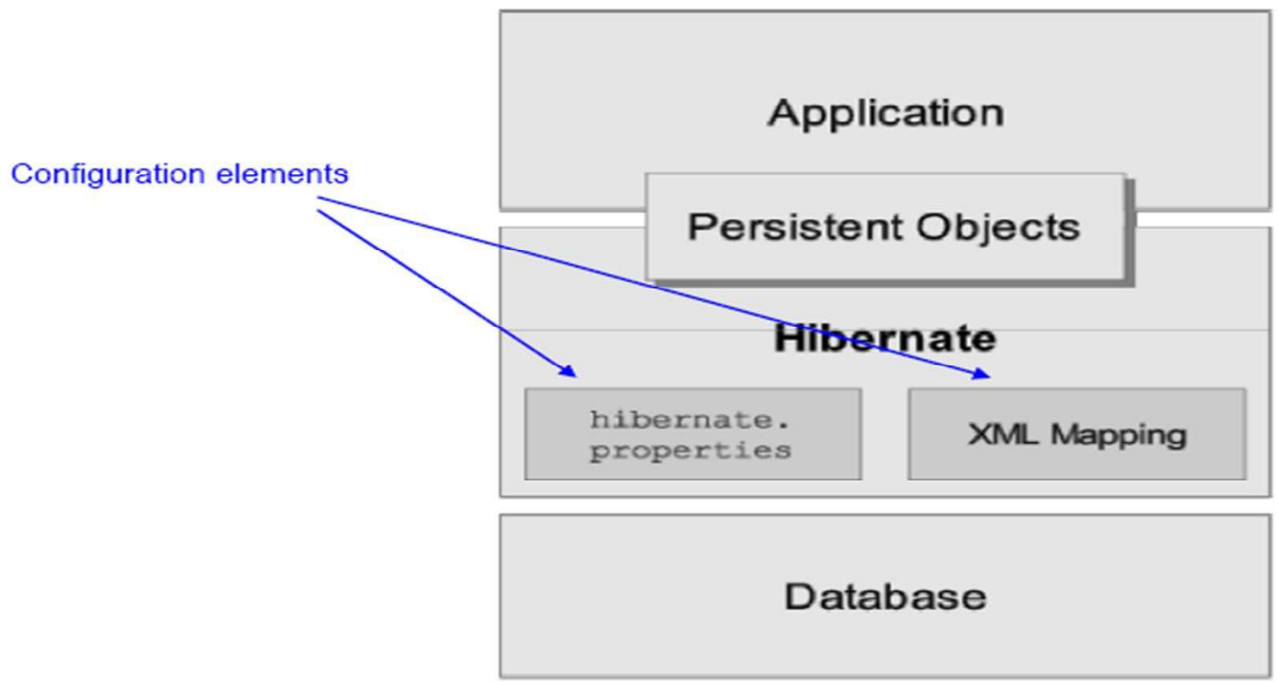
Configuring Hibernate

Objectives

At the end of this lesson, you will be able to:

- Illustration of Hibernate Samples
- Creating a simple, but full, end to end Hibernate Application
- Introduction to JUnit

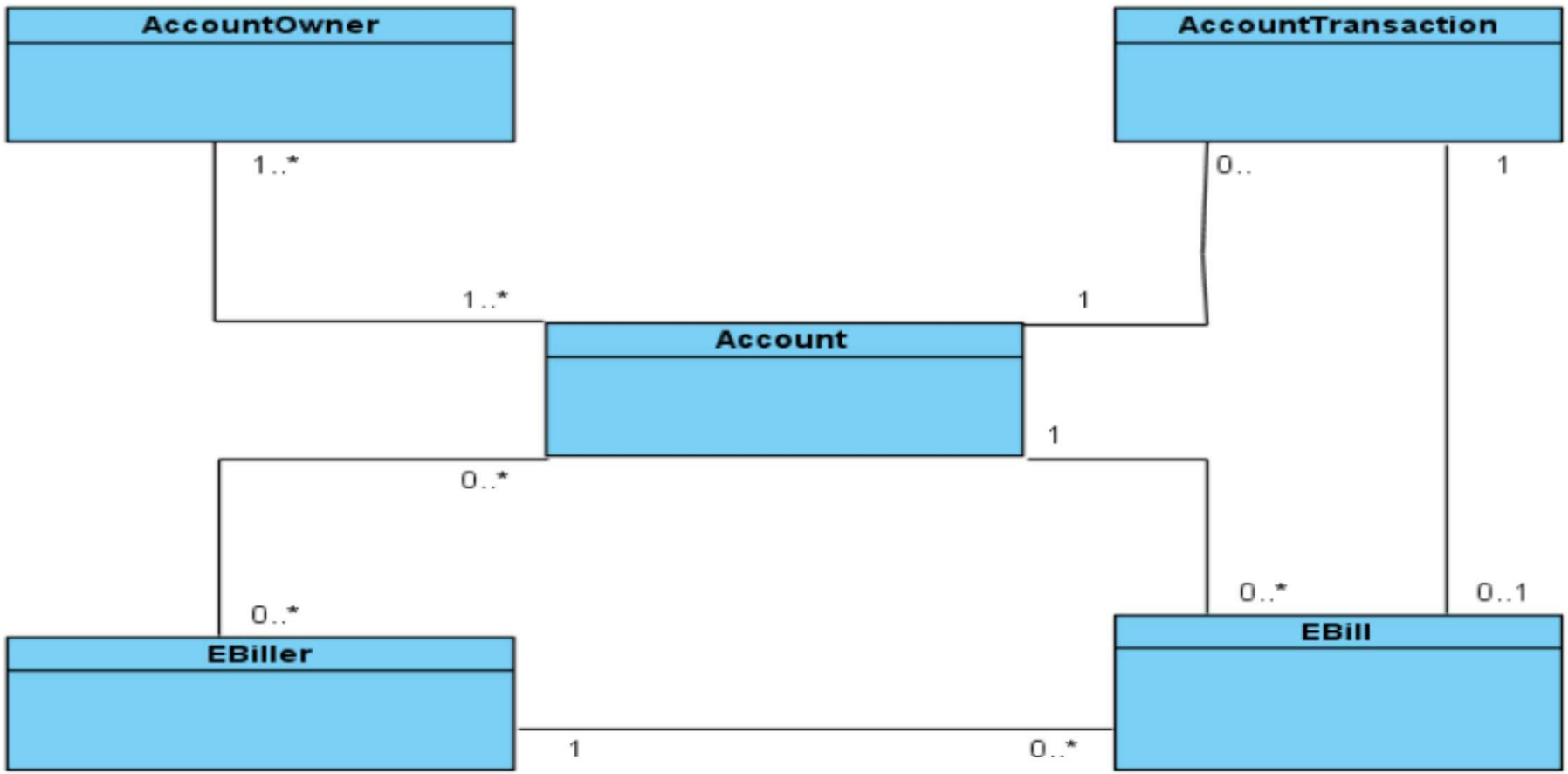
High Level Architecture



Building a Hibernate Application

1. **Define the domain model**
2. **Setup your Hibernate configuration**
   - hibernate.properties / 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**

Problem Domain – eBank



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">
```

All Hibernate configuration files validate against their appropriate xml DTDs

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

Configure Hibernate here, particularly the session-factory

hibernate.cfg.xml

```
...
<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

```
    ...

    <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

```
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

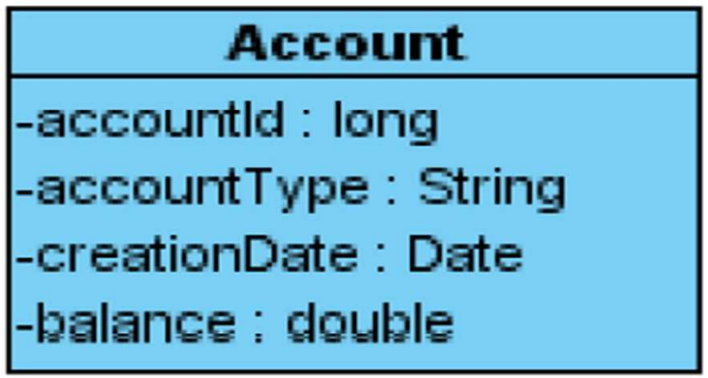
```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/
   hibernate-mapping-3.0.dtd">
```

DTD for the object mapping files

```
<hibernate-mapping>
  <class>
  . . .
  </class>
</hibernate-mapping>
```

Describe your class attributes here.

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	-	-

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>
```

Mapping file named after Java Object

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…**

Identify Mapping Files in the

```
    ...

    <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>
```

Tell Hiberate about the mapped objects!

- **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

```
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;
   }
}
```

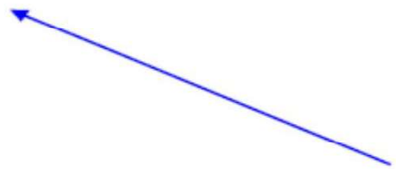
Recommended to catch Throwable for initialization error!

Common Methods of Session API

- **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()

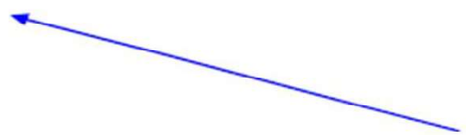
```
public void saveOrUpdateAccount(Account account) {

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

   session.saveOrUpdate(account);

}
```



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

JDBC Example – Create Account

```
public Account createAccount(Account account) {
  Connection connection = null;
  PreparedStatement getAccountIdStatement = null;
  PreparedStatement createAccountStatement = null;
  ResultSet resultSet = null;
  long accountId=0;

  try {
    Connection connection = DriverManager.getConnection("jdbc:oracle:thin:lecture1/password@localhost:1521:XE");
    connection.setAutoCommit(false);
    getAccountIdStatement = connection.prepareStatement("SELECT ACCOUNT_ID_SEQ.NEXTVAL FROM DUAL");
    resultSet = getAccountIdStatement.executeQuery();
    resultSet.next();
    accountId = resultSet.getLong(1);

    createAccountStatement = connection.prepareStatement(AccountDAOConstants.CREATE_ACCOUNT);
    createAccountStatement.setLong(1, accountId);
    createAccountStatement.setString(2,account.getAccountType());
    createAccountStatement.setDouble(3, account.getBalance());
    createAccountStatement.execute();

    connection.commit();
  }
  catch (SQLException e) {
    try{
      connection.rollback();
    }catch(SQLException e1){// log error}
      throw new RuntimeException(e);
    }
  finally {
  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}
  }
```



Approx 37 lines of code (not counting loading the SQL Driver and SQL statement constants!)

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);

}
```

Testing with JUnit

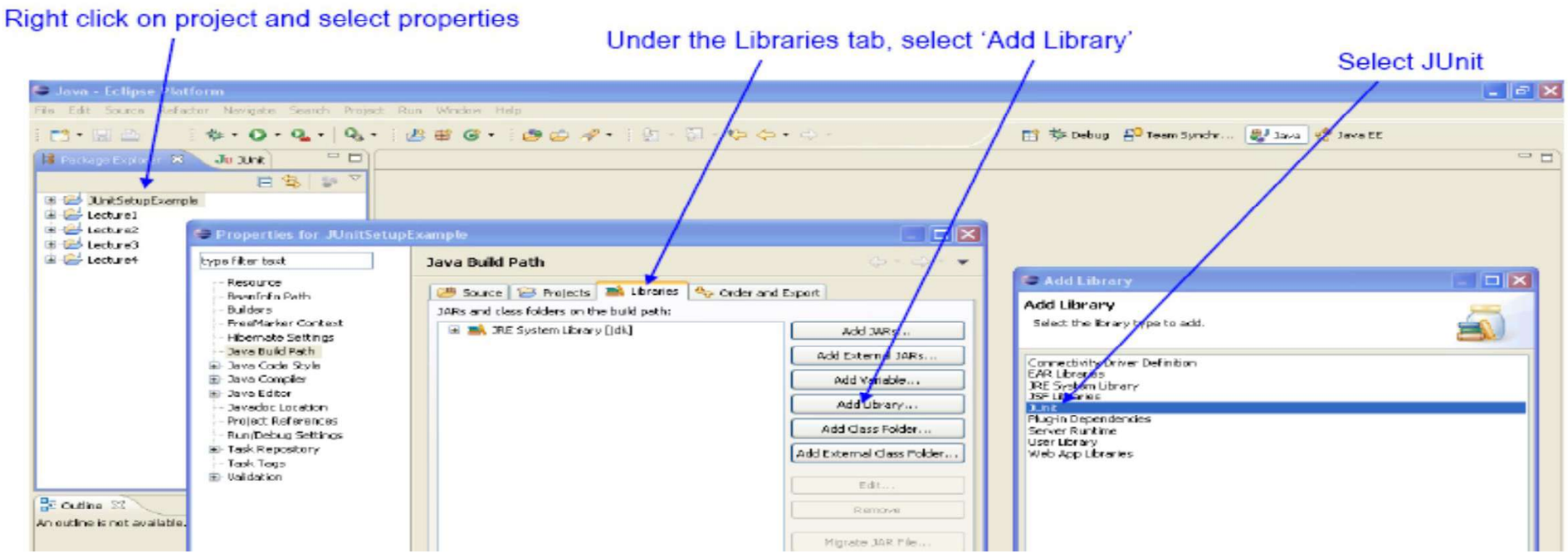
- **JUnit is an open source framework to perform testing against units of code.**
  - A single test class contains several test methods
  - Provides helper methods to make ‘assertions’ of expected results
  - Common to have multiple test classes for an application

Using JUnit

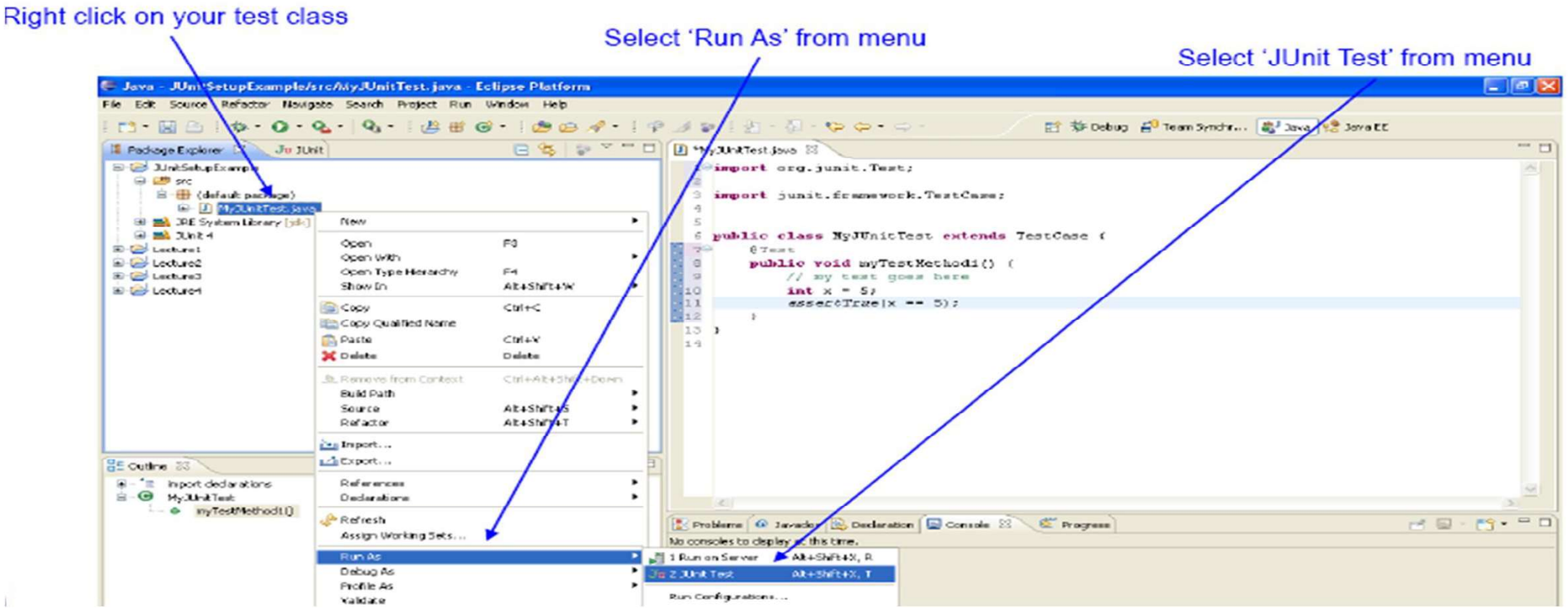
- 1. Download the jar from JUnit.org
- 2. Add downloaded jar to project classpath
- 3. Create a class to house your test methods, naming it anything you ` like (typically identifying it as a test class)
- 4. Implement test methods, naming them anything you like and marking each with the **@Test** annotation at the method level
- 5. 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**



- **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

```
    ...

    // 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);

}
```

- **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 Create – Output

```
Hibernate: select hibernate_sequence.nextval from dual

Hibernate: insert into ACCOUNT (CREATION_DATE,
    ACCOUNT_TYPE, BALANCE, ACCOUNT_ID) values (?, ?, ?, ?)

1453 [main] INFO org.hibernate.impl.SessionFactoryImpl - closing
1453 [main] INFO
    org.hibernate.connection.DriverManagerConnectionProvider - cleaning
    up connection pool: jdbc:oracle:thin:@localhost:1521:XE


var account =
----ACCOUNT----
accountId=1
accountType=SAVINGS
creationDate=Sat Sep 13 21:53:01 EDT 2008
balance=1000.0
----ACCOUNT----
```

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());

   System.out.println(account);

   // make sure these are two separate instances
   Assert.assertTrue(account != anotherCopy);
   System.out.println("var anotherCopy = "
        + anotherCopy);

   session.getTransaction().commit();
   HibernateUtil.getSessionFactory().close();
}
```

Test Get - Output

```
var account =
----ACCOUNT----
accountId=21
accountType=SAVINGS
creationDate=Sat Sep 13 22:54:00 EDT 2008
balance=1000.0
----ACCOUNT----

Hibernate: select account0_.ACCOUNT_ID as ACCOUNT1_0_0_,
    account0_.CREATION_DATE as CREATION2_0_0_,
    account0_.ACCOUNT_TYPE as ACCOUNT3_0_0_, account0_.BALANCE
    as BALANCE0_0_ from ACCOUNT account0_ where
    account0_.ACCOUNT_ID=?

 var anotherCopy =
 ----ACCOUNT----
 accountId=21
 accountType=SAVINGS
 creationDate=2008-09-13 22:54:00.0
 balance=1000.0
 ----ACCOUNT----
```

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();

   ...
```

```
    ...

    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 Update Balance - Output

```
Hibernate: update ACCOUNT set BALANCE=? where ACCOUNT_ID=?

Hibernate: select account0_.ACCOUNT_ID as ACCOUNT1_0_0_,
    account0_.CREATION_DATE as CREATION2_0_0_,
    account0_.ACCOUNT_TYPE as ACCOUNT3_0_0_,
    account0_.BALANCE as BALANCE0_0_ from ACCOUNT
    account0_ where account0_.ACCOUNT_ID=?

var anotherCopy =
----ACCOUNT----
accountId=22
accountType=SAVINGS
creationDate=2008-09-13 22:56:42.296
balance=2000.0
----ACCOUNT----
```

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();

   ...
```

```
...

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();

}
```

Test Delete - Output

```
Hibernate: delete from ACCOUNT where ACCOUNT_ID=?

Hibernate: select account0_.ACCOUNT_ID as ACCOUNT1_0_0_,
    account0_.CREATION_DATE as CREATION2_0_0_,
    acacount0_.ACCOUNT_TYPE as ACCOUNT3_0_0_,
    account0_.BALANCE as BALANCE0_0_ from ACCOUNT
    account0_ where account0_.ACCOUNT_ID=?

var anotherCopy = null
```

Java Class Attribute Type	Hibernate Type	Possible SQL Type—Vendor Specific
Integer, int, long short	integer, long, short	Appropriate SQL type
char	character	char
java.math.BigDecimal	big_decimal	NUMERIC, NUMBER
float, double	float, double	float, double
java.lang.Boolean, boolean	boolean, yes_no, true_false	boolean, int
java.lang.string	string	varchar, varchar2
Very long strings	text	CLOB, TEXT
java.util.Date	date, time, timestamp	DATE, TIME, TIMESTAMP
java.util.Calendar	calendar, calendar_date	TIMESTAMP, DATE
java.util.Locale	locale	varchar, varchar2
java.util.TimeZone	timezone	varchar, varchar2
java.util.Currency	Currency	varchar, varchar2
java.sql.Clob	clob	CLOB
java.sql.Blob	blob	BLOB
Java object	serializable	binary field
byte array	binary	binary field
java.lang.Class	class	varchar, varchar2

Summary

In this lesson, you have learned

- Illustration of Hibernate Samples
- Creating a simple, but full, end to end Hibernate Application
- Introduction to JUnit