Object-Relational Mapping (ORM)

Course Outline

- **Introduction to ORM**
  - What is JDBC?
  - Why ORM?
  - What is ORM?
  - Java ORM/Persistent Frameworks
- **Hibernate Overview**
  - What is Hibernate?
  - Why Hibernate?
  - Supporting Database Management Systems Engine
- **Hibernate Architecture**
  - Hibernate Architecture and API

Course Outline (Ex.)

- **Hibernate Configuration**
  - Hibernate Installation/Setup
  - Configuration Properties
  - Mapping File(s)
  - Development Strategies
- **First Example (Hello Hibernate)**
- **Hibernate Entity Life-Cycle**
  - Entity Life-Cycle (Object States)
  - Session Operations

Course Outline (Ex.)

- **Entities Associations**
  - Many-to-one (Uni-directional and Bi-directional)
  - One-to-one (Uni-directional and Bi-directional)
  - Many-to-many (Uni-directional and Bi-directional)
  - Inheritance Mapping Strategies
  - Table per concrete classes
  - Table per subclasses
  - Shared Table per concrete class with unions
  - Table per joined subclasses

Course Outline (Ex.)

- **Transitive Persistence**
  - Lazy and Eager loading
  - Fetching Strategies
    - Join Fetching Strategy
    - Select Fetching Strategy
    - Sub-Select Fetching Strategy
    - Batch Size Fetching Strategy
  - The Second Level Cache
    - Read-Only Cache
    - Read-Write Cache
    - Nonstrict-Read-Write Cache

Course Outline (Ex.)

- **Hibernate Query (by HQL)**
  - Hibernate Query
  - HQL Parameter Binding
  - HQL Restrictions
  - HQL Comparison Expressions
  - Standard HQL Functions
  - Added HQL Functions
  - HQL Ordering
  - HQL Projections
  - HQL Join(s)
  - HQL Grouping & Group Restrictions
  - HQL Dynamic Instantiation
  - HQL Sub-Queries

Course Outline (Ex.)

- **Hibernate Query (by Criteria)**
  - Parameter Binding
  - Restrictions
  - Comparison Expressions
  - String Matching
  - Logical Operators
  - Sub-Queries
  - Join(s)
  - Projections
  - Aggregation and Grouping & Group Restrictions
  - Query By Example

Course Outline (Ex.)

- **Advanced Topics**
  - Interceptor
  - Event Listeners
  - Naming Strategy

Introduction to ORM

Lesson Outline

- What is JDBC?
- Why ORM?
- What is ORM?
- Java ORM/Persistent Frameworks

Lesson Outline

- What is JDBC?
- Why ORM?
- What is ORM?
- Java ORM/Persistent Frameworks

What is JDBC?

- Stand for Java Database Connectivity
- Set of Java API for accessing the relational databases from Java program.
  - Java API that enables Java programs to execute SQL statements and interact with any SQL-compliant database.
- Possible to write a single database application that can run on different platforms and interact with different DBMS.

Pros and Cons of JDBC

- **Pros**
  - Clean and simple SQL processing
  - Good performance with large amounts of data.
  - Very good for small applications

JDBC Pros and Cons. (Ex.)

- **Cons**
  - Large programming overhead in large projects.
  - Transactions and concurrency must be hand-coded
  - Handling the JDBC connections and closing the connection is also a big issue
  - No encapsulation
  - Hard to maintain
  - Query is DBMS specific
  - Hard to implement MVC concept

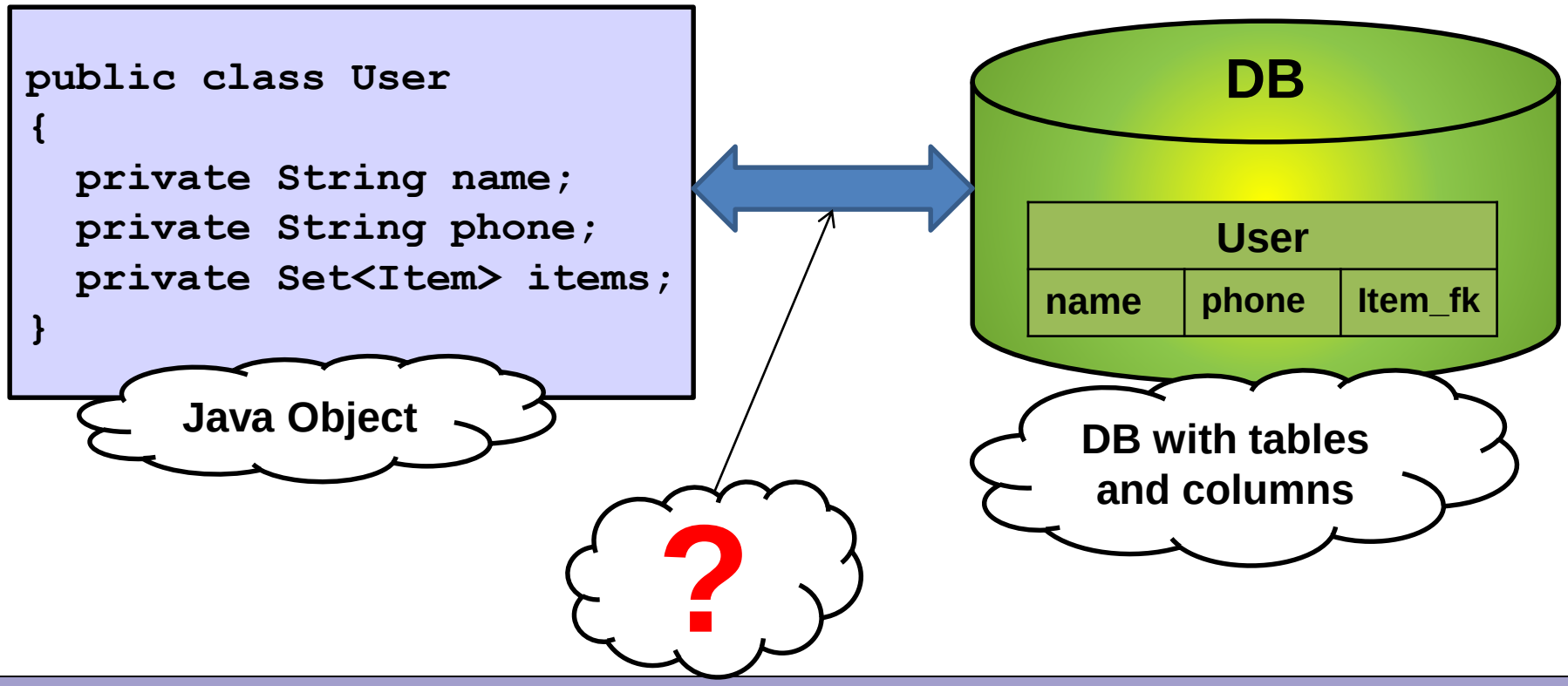
Lesson Outline

- What is JDBC?
- Why ORM?
- What is ORM?
- Java ORM/Persistent Frameworks

Why ORM ? (Problem Area)

- When working with object-oriented systems,
  - There's a mismatch between

    The object model & the relational database.



Problem Area (Ex.)

```
public void addUser( User user )
{
   String sql = "INSERT INTO user
              (name,address)
             VALUES('" + user.getName()+ "','"
             + user.getAddress() + "')";

      // Initiate a Connection,
      // create a Statement,
      // and execute the statement
}
```

Problem Area (Ex.)

```
public User getUser( User user )
{
   String sql = "select * from user where
            name = '" + user.getName() +
            "' and age > " + user.getAge();

      // Initiate a Connection,
      // create a Statement,
      // and execute the query

      return user;
}
```

Problem Area (Ex.)

- Write SQL conversion methods by hand using JDBC:
  - Tedious and requires lots of code
  - Extremely error-prone
  - Non-standard SQL ties the application to specific databases
  - Difficult to represent associations between objects

The preferred solution

- Use a Object-Relational Mapping (ORM) System.
- Provides a simple API for storing and retrieving Java objects directly to and from the database.
- **Non-intrusive:** No need to follow specific rules or design patterns.
- **Transparent:** Your object model is unaware

The preferred solution

Lesson Outline

- What is JDBC?
- Why ORM?
- What is ORM?
- Java ORM/Persistent Frameworks

What is ORM ?

- **Object-Relational Mapping (ORM)**
  - is a programming technique for converting data between relational databases and object-oriented programming languages.
- Lets business code access objects rather than DB tables
- Hides details of SQL queries from OO logic
- Based on JDBC ‘under the hood’

ORM Pros

- No need to deal with the database implementation
  - Little need to write SQL statements.
- Entities based on business concepts rather than database structure
- Fast development of application
- Transaction management and automatic key generation.

Lesson Outline

- What is JDBC?
- Why ORM?
- What is ORM?
- Java ORM/Persistent Frameworks

Java ORM/Persistent Frameworks

- More than 23 Implementation for ORM.
- Some of the most common ORM Frameworks:
  - Hibernate, open source ORM framework, widely used
  - Java Persistence API (JPA), Standard
  - TopLink by Oracle
  - EclipseLink, Eclipse persistence platform
  - iBATIS(MyBatis), maintained by ASF
  - Enterprise Objects Framework, Mac OS X/Java, part of Apple WebObjects

Hibernate Overview

Lesson Outline

- What is Hibernate?
- Why Hibernate?
- Supporting Database Management Systems Engine

Lesson Outline

- What is Hibernate?
- Why Hibernate?
- Supporting Database Management Systems Engine

What is Hibernate?

- Hibernate is an ORM solution for JAVA.
- It is a powerful, high performance object/relational persistence and query service.
- It allows us to develop persistent classes following object-oriented idiom – including association, inheritance and polymorphism.

Lesson Outline

- What is Hibernate?
- Why Hibernate?
- Supporting Database Management Systems Engine

Why Hibernate?

- Hibernate maps Java classes to database tables using XML files or annotations
  - No need to write code for this.
  - If there is change in Database or in any table then the only need to change XML file properties.

- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.

Why Hibernate? (Ex.)

- Manipulates Complex associations of objects of your database.
- Provides Simple querying of data.
- Minimize database access with smart fetching strategies.
- Caching.
- Easy transaction handling.

Lesson Outline

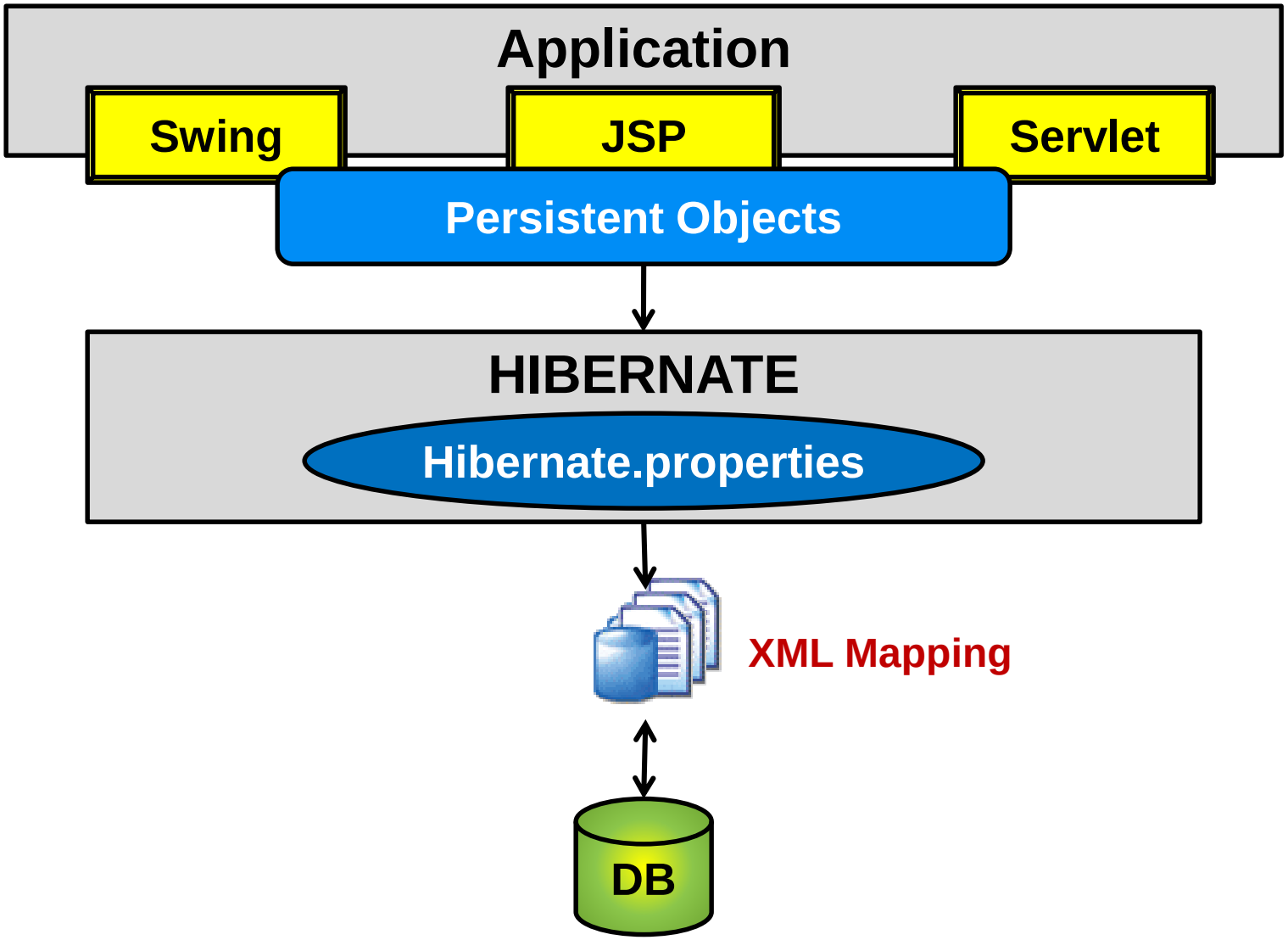
- What is Hibernate?
- Why Hibernate?
- Supporting Database Management Systems Engine

Supporting Database Engines

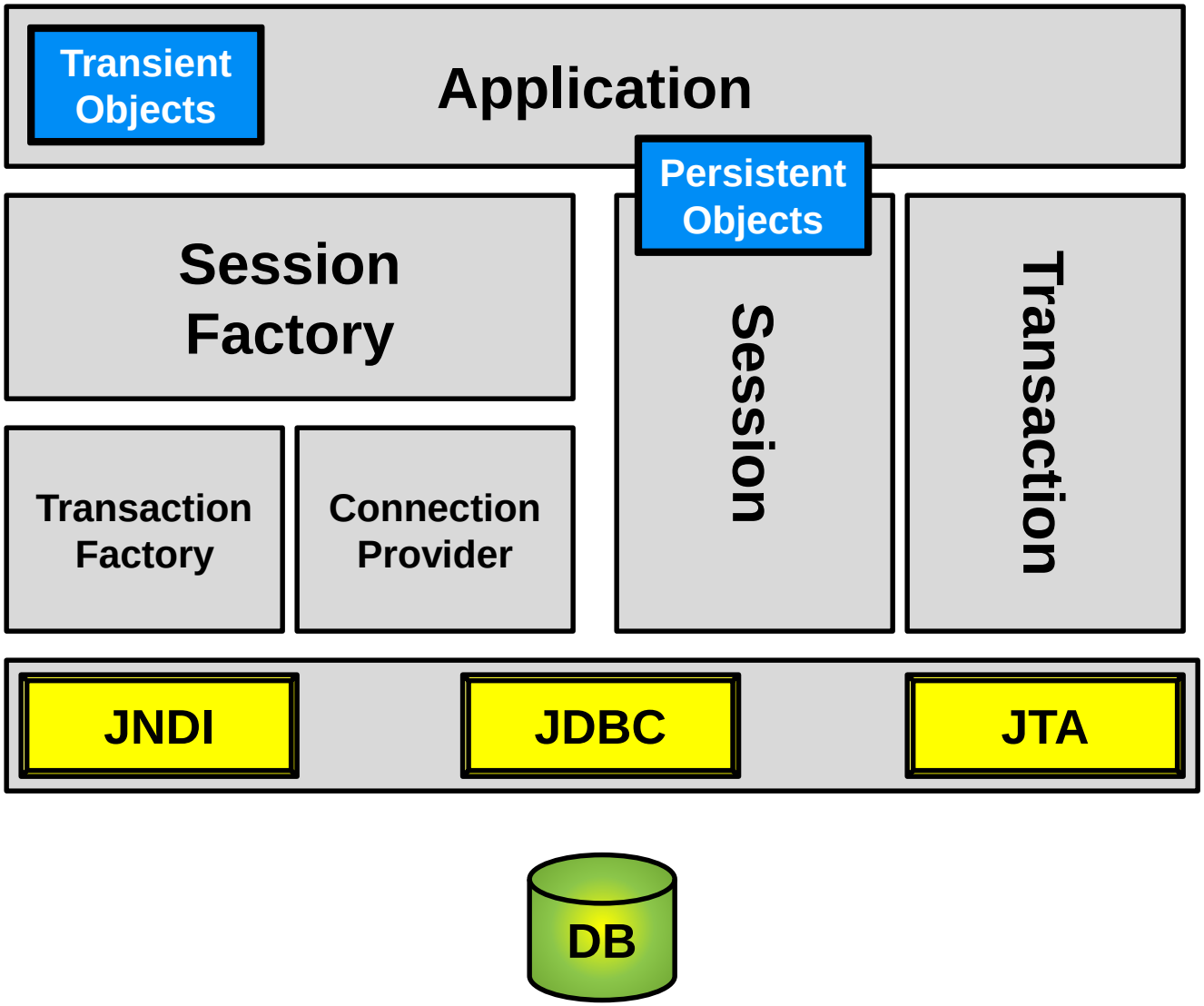
- Support more than 27 different DBMS with different base (dependant, XML)
- Some of most common DBMS:
  - **Oracle**
  - **DB2**
  - **Microsoft SQL Server**
  - **Sybase**
  - **MySQL**
  - **SAP DB**

Hibernate Architecture

Hibernate Architecture "High Level"



Hibernate Architecture "Low Level"



Hibernate Architecture

- **Session Factory :**
  - Caches mappings for a single database.
  - A factory for Session.
  - to Create the SessionFactory from hibernate.cfg.xml

```
SessionFactory fact = new Configuration()
        .configure("/hibernate.cfg.xml")
        .buildSessionFactory();
```

Hibernate Architecture

- **Session Factory :**
  - Caches mappings for a single database.
  - A factory for Session.
  - to Create the SessionFactory from hibernate.cfg.xml

```
ServiceRegistry standardRegistry =
new StandardServiceRegistryBuilder()
                .configure("/hibernate.cfg.xml")
                .build();

Metadata metadata =
new MetadataSources(standardRegistry).buildMetadata();
SessionFactory fact =
metadata.getSessionFactoryBuilder().build();
```

Hibernate Architecture (Ex.)

- **Session:**
  - A single-threaded, short-lived object representing a conversation between the application and the persistent.
  - A wrapper for the JDBC Connection.
  - A factory for Transaction
    - sessionFactory.openSession();
    - sessionFactory.getCurrentSession();

Hibernate Architecture (Ex.)

- **Transaction:**
  - Specifies atomic units of work on DB Level.
- **ConnectionProvider:**
  - A factory for (and pool of) JDBC connections
  - It is NOT exposed to the application level.
- **TransactionFactory:**
  - A factory for Transaction instances.
  - It is NOT exposed to the application level.

Hibernate Configuration

Lesson Outline

- Hibernate Installation/Setup
- Configuration Properties
- Mapping File(s)
- Development Strategies

Lesson Outline

- Hibernate Installation/Setup
- Configuration Properties
- Mapping File(s)
- Development Strategies

Hibernate Installation/Setup

- As we know hibernate is pluggable framework so to use hibernate in your application:
  - Create your project as you want (Desktop, Web, Enterprise)
  1. Add the Hibernate Core Lib
  2. Add the hibernate dependencies (mandatory for some versions)
  - However after this steps your application is ready to run hibernate, but for more usability you must configure your IDE to support Hibernate tools like eclipse.
  - Netbeans automatic enable HTools.

Hibernate Installation/Setup

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.4.29.Final</version>
 </dependency>
```

Lesson Outline

- Hibernate Installation/Setup
- Configuration Properties
- Mapping File(s)
- Development Strategies

Hibernate Configuration file

- **XML configuration file:**
  - Specify a full configuration in a file with standard name " `hibernate.cfg.xml` ".
  - The most important configuration parameters are:
    - Database name
    - Database dialect (type)
    - Database credentials
    - Database source name
    - Hibernate mapping files.
    - Caching settings
    - Connection Pooling settings

Hibernate Configuration file (Ex.)

```
<hibernate-configuration>
   <session-factory>
      <property name = "hibernate.connection.driver_class">
      com.mysql.cj.jdbc.Driver</property>
      <property name = "hibernate.connection.url">
         jdbc:mysql://localhost:3306/helloworlddb
      </property>
      <property name = "hibernate.connection.username">
         root</property>
      <property name ="hibernate.connection.password">
         root</property>
      <property name ="hibernate.dialect">
         org.hibernate.dialect.MySQL5Dialect</property>
     <mapping resource="dao/Person.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Lesson Outline

- Hibernate Installation/Setup
- Configuration Properties
- Mapping File(s)
- Development Strategies

Hibernate Mapping file

- XML documents that defines
  - the mapping between the java class DAO and the correspondence database table.
- Can be generated from the Java source , database using specific tools

Hibernate Mapping file (Ex.)

- It Defines the mapping of these to the database.
  - Mapping Classes to Tables
  - Mapping Properties to Columns
  - Mapping Id field
  - Generating object identifiers
  - Key Generation approach.
  - object relationships types (one to one, one to many ,…)
  - Fetching strategies

Hibernate Mapping file (General Form)

```
<hibernate-mapping>
   <class name="class-name" table="corresponding-table-
     name">
     <id name="attribute-name" column="column-name"
        type="hibernate-type" >
        <generator class="generator-class"/>
     </id>
     <property name="property-name" column="column-name"
               type="hibernate-type"/>
     <property name="property-name" column="column-name"
               type="hibernate-type"/>
   </class>
</hibernate-mapping>
```

Hibernate Mapping file (Example)

```
<hibernate-mapping>
    <class name="dao.Person" table="person"
      catalog="helloorm">
      <id name="id" column="id" type="int" >
         <generator class="identity"/>
      </id>
      <property name="name" column="name" type="string"/>
      <property name="address" column="address"
       type="string"/>
      <property name="phone" column="phone" type="string"/>
      <property name="email" column="email" type="string"/>
      <property name="birthday" column="birthday"
       type="date"/>
    </class>
</hibernate-mapping>
```

Generator

- The optional **<generator>** element used to generate unique identifiers for instances of the persistent class.

- There are built-in generators :

  1. Increment
  2. Identity
  3. Sequence
  4. Native

Generator

- **increment**
  - generates identifiers of type **long**, **short** or **int** that are unique only when no other process is inserting data into the same table.
  - *Do not use in a cluster.*

- **identity**
  - supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL.
  - The returned identifier is of type **long**, **short** or **int**.

Generator

- **sequence**
  - uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, McKoi or a generator in Interbase.
  - The returned identifier is of type **long**, **short** or **int**

- **native**
  - picks identity, sequence or hilo depending upon the capabilities of the underlying database.

Custom Generators

- All generators implement the interface **`org.hibernate.id.IdentifierGenerator`**

- some applications may choose to provide their own specialized implementations by implement **`IdentifierGenerator`** interface.

Annotations

- is a special form of syntactic metadata that can be added to Java source code.
  - Classes.
  - Methods.
  - Variables.
  - Parameters.
  - Packages.
  - Annotations
- Java annotations can be reflective:
  - They can be embedded in class files generated by the compiler and may be retained by the Java VM or other framework to be made retrievable at run-time

Hibernate Annotations

- Available from Hibernate 3.5
- Basic annotations that implement the JPA standard:

• @Entity	Declares this an entity bean
• @Id • @EmbeddedId • @GeneratedValue	Identity
• @Table • @Column	Database Schema Attributes
• @OneToOne • @ManyToOne • @OneToMany	Relationship mappings

Hibernate Annotations (Ex.)

- Hibernate extension annotations:
  - Contained in org.hibernate.annotations package
    - @org.hibernate.annotations.Entity
    - @org.hibernate.annotations.Table
    - @BatchSize
    - @Where
    - @Check

Hibernate Annotations (Ex.)

- Old Technique:
  - Hibernate configuration file.
  - Hibernate Mapping Files.
  - Hibernate Mapping Classes (Entities).
  - Hello with Login Example.
- New Technique (+ Annotation):
  - Hibernate configuration file.
  - Hibernate Mapping Classes (Entities).
  - Hello with Login Example.

Hibernate Annotations (Ex.)

```
<hibernate-configuration>
   <session-factory>
      <property name = "hibernate.connection.driver_class">
         com.mysql.cj.jdbc.Driver</property>
      <property name = "hibernate.connection.url">
         jdbc:mysql://localhost:3306/helloworlddb
      </property>
      <property name = "hibernate.connection.username">
         root</property>
      <property name ="hibernate.connection.password">
         root</property>
      <property name ="hibernate.dialect">
         org.hibernate.dialect.MySQL5Dialect
      </property>
  <mapping resource="dao/Person.hbm.xml" class="dao.Person"/>
  </session-factory>
</hibernate-configuration>
```

Hibernate Annotations (Ex.)

```
@Entity
@Table(name = "account")
public class Account{
    private long id;
    private String userName;

     @Id
     @GeneratedValue(strategy = GenerationType.IDENTITY)
     @Column(name = "id")
     public long getId() {      return id;   }
     @Basic(optional =false )
     @Column(name = "user_name")
     public String getUserName() {    return userName;    }
}
```

Account.java

Hibernate Annotations (Ex.)

```
@Entity
@Table(name = "account")
public class Account implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "user_name")
    private String userName;
    @Temporal(TemporalType.TIMESTAMP)
    private Date birthday;……}
```

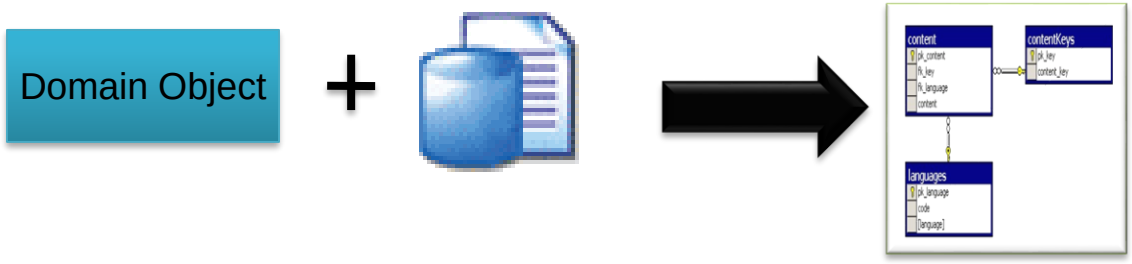
Account.java

Lesson Outline

- Hibernate Installation/Setup
- Configuration Properties
- Mapping File(s)
- Development Strategies

Development Strategies

- Selecting a development Strategy :
  - **Top down**
    - Create your java domain object,
    - Create xml mapping files and
    - Generate the database schema.



Development Strategies (Ex.)

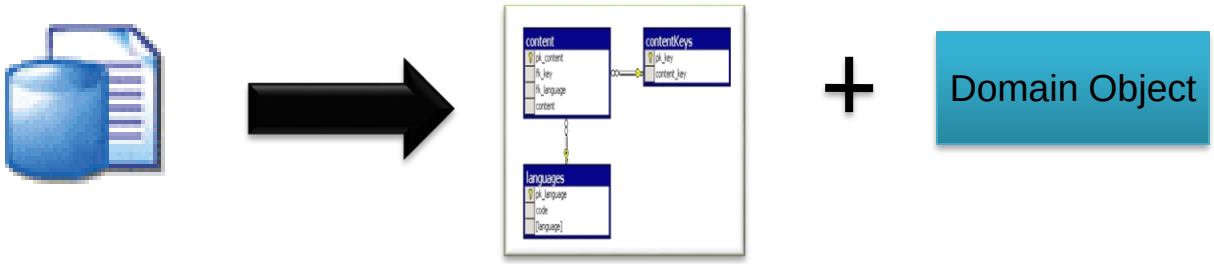
– **Bottom up**

- Create database and then
- Generate the xml mapping files and java domain objects.

Development Strategies (Ex.)

– **Middle out**

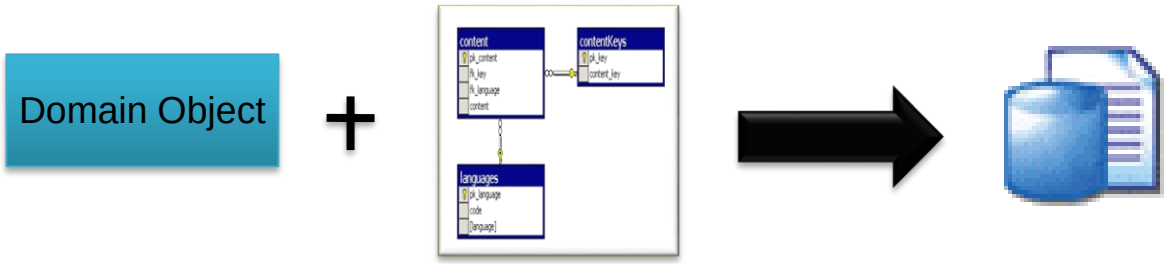
- Write xml mapping files and then
- Generate java domain objects and database



Development Strategies (Ex.)

– **Meet in the middle**

- You already have java classes and database.
- This scenario usually requires at least some re-factoring of the Java classes, database schema, or both.



First Example
(Hello Hibernate)

Hello Hibernate

- Steps to create your first application using Hibernate:
  - Create your java project
  - Add the required jars to your project
  - Create a java bean that'll represent the corresponding table
  - Create the XML mapping file which should be saved as `className.hbm.xml` and located near the class
  - Create `hibernate.cfg.xml` to configure the Hibernate
  - Create a test class to insert object from the DB.

Hello Hibernate (Ex.)

```
public class Account{
    private int id;
    private String userName;
    private String fullName;
    private String phone;
    private String address;
    private String password;
    private Date birthday;

    public Account()
    {   }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    // getter & setter for all attributes
}
```

Account.java

Hello Hibernate (Ex.)

```
<hibernate-mapping>
    <class name="dao.Account" table="account"
      catalog="helloworlddb">
      <id name="id" column="id" type="int" >
         <generator class="identity"/>
      </id>
      <property name="userName" column="user_name"
type="string"/>
      <property name="fullName" column="full_name"/>
      <property name="password" column="password"/>
      <property name="address" column="address"/>
      <property name="phone" column="phone" type="string"/>
      <property name="birthday" column="birthday"
       type="date"/>
    </class>
</hibernate-mapping>
```

Account.hbm.xml

Hello Hibernate (Ex.)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "...">
<hibernate-configuration>
   <session-factory>
      <property name = "hibernate.connection.driver_class">
         org.gjt.mm.mysql.Driver                    </property>
      <property name = "hibernate.connection.url">
        jdbc:mysql://localhost:3306/helloworlddb </property>
      <property name = "hibernate.connection.username">
         root                                       </property>
      <property name ="hibernate.connection.password">
         root                                       </property>
      <property name ="hibernate.dialect">
         org.hibernate.dialect.MySQLInnoDBDialect
      </property>
     <mapping resource="dao/Account.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

hibernate.cfg.xml

Hello Hibernate (Ex.)

```
public class Test {
    public static void main(String[] args) {
        SessionFactory sessionFactory = new Configuration()
            .configure().buildSessionFactory();
        Session session = sessionFactory.openSession();

        Account account = new Account();
        account.setName("Medhat");
        account.setPhone("0235355637");
        account.setBirthday(new Date());
        account.setEmail("ahyousif@mcit.gov.eg");

        session.beginTransaction();
        session.persist(account);
        session.getTransaction().commit();
        System.out.println("Insertion Done");
    }
}
```

Test.java