**Project Management - Git**

1. **Github and Bitbucket**

Visit <https://github.com/>

Visit <https://bitbucket.org/product>

1. **Installation Git**

Visit <https://git-scm.com/downloads> and download latest version of Git for Windows and install follow the instruction step by step

1. **Basic command line for Git**

*git checkout*

*git branch*

*git fetch*

*git pull*

*git push*

*git add*

*git merge*

*git log*

*git stash*

*git revert*

*git cherry-pick*

…

1. **Installation SourceTree**

Visit <https://www.sourcetreeapp.com/> and download latest version of SourceTree for Windows and install follow the instruction step by step

1. **Git Integration for Eclipse**

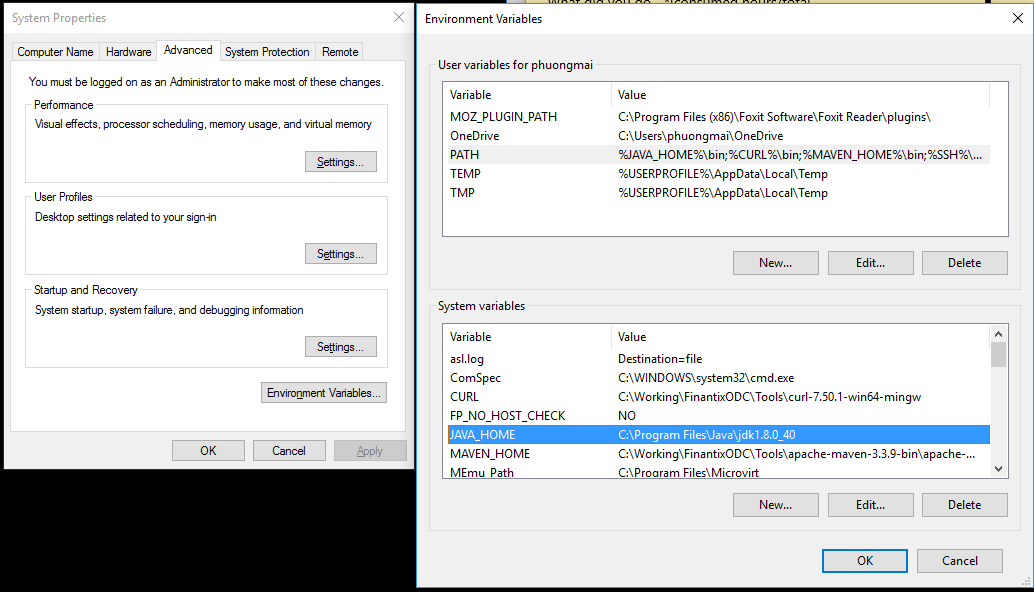
Open Eclipse -> **Help** -> **Eclipse Marketplace** -> Search **EGit** keyword -> Click **Install** button at **EGit - Git Integration for Eclipse** section and follow the instruction step by step

**Apache Maven**

1. **Maven is a tool with multiple facets:**

* A **Build** tool – Generated artifacts (jar, war, ear, …)
* A **Dependency Management** tool
* A **Documentation** tool – Generated test results, quality metrics, Javadoc, …

1. **Require**

Make sure JDK is installed and JAVA\_HOME variable is added

1. **Installation:**

Visit <http://maven.apache.org/download.cgi> and download Maven zip file ***apache-maven-3.5.0-bin.zip***

Unzip the file in a local directory such as*:* ***C:\maven***

Set environment variables:

***MAVEN\_HOME=C:\maven***

***PATH=%MAVEN\_HOME%\bin***

1. **Verification**

Open cmd and enter **mvn – version** in the command line

1. **Maven eclipse plugin (for eclipse old version)**
2. Open Eclipse -> Click **Help** -> **Install New Software** -> Click **Add** button at top corner and fill up

***Name=Maven*** *and* ***Location=***[***http://download.eclipse.org/technology/m2e/releases/***](http://download.eclipse.org/technology/m2e/releases/)

A **check-box** will appear in the pop window, **Check** and click **Next** button and follow the instruction step by step

Another way to install Maven plugin:

1. Open Eclipse -> **Help** -> **Eclipse Marketplace** -> Search **Maven** keyword -> Click **Install** button at **Maven Integration for Eclipse** section and follow the instruction step by step
2. **Maven command structure**

**mvn eclipse:eclipse** //generated maven project for Eclipse IDE

**mvn clean**

**mvn clean install** // install the artifact in your local respository(${user.home}/.m2/repository)

**mvn compile**

**mvn package**

**mvn test**

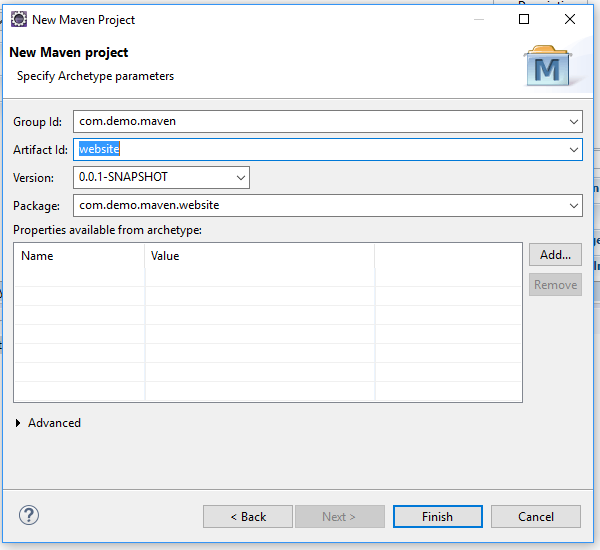
…

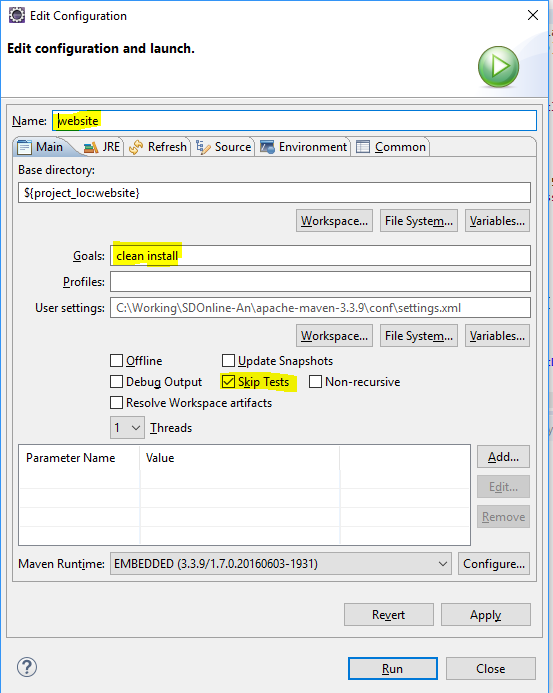
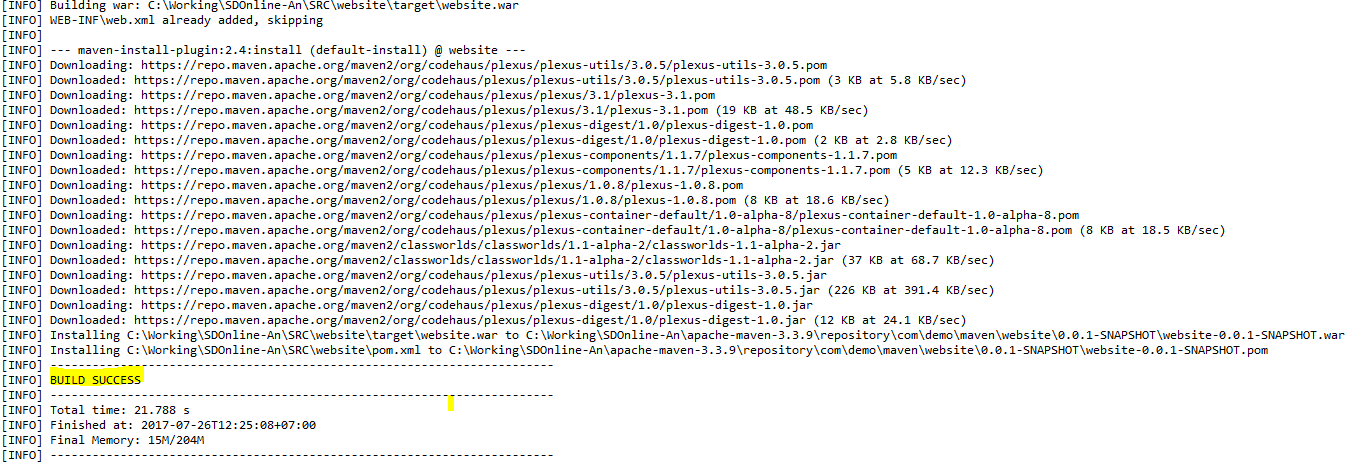
1. **Create Maven project using Eclipse IDE**

Open Eclipse -> File -> New -> **Maven Project**

Select default Workspace location

Select the maven archetype as: **maven-archetype-webapp**

Fill out details and click Finish

Now build project with **maven clean install** to check dependency issues with project and deploy application on Apache Tomcat Server

**Apache Tomcat Server**

1. **Installation**

Visit <https://tomcat.apache.org/download-70.cgi> and download Apache tomcat server zip file

Upzip the file in local directory such as: ***C:\tomcat***

1. **Create tomcat server on Eclipse**
2. **Deploy project into tomcat server on Eclipse**

**Hibernate with MySQL**

1. **Create maven project from Eclipse IDE**

*(Review last session)*

1. **Add Hibernate and Mysql dependency**

Hibernate is required ***dom4j, commons-logging, commons-collections and cglib*** as dependency library

In case using Anotation, it’s required to download the Hibernate annotation library ***hibernate-annotations and hibernate-commons-annotations and repository.jboss***

Modify the ***pom.xml*** file.

|  |
| --- |
| ...  <repositories>  <repository>  <id>JBoss repository</id>  <url>http://repository.jboss.com/maven2/</url>  </repository>  </repositories>  <dependencies>  <!-- MySQL database driver -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.9</version>  </dependency>  <!-- Hibernate core -->  <dependency>  <groupId>hibernate</groupId>  <artifactId>hibernate3</artifactId>  <version>3.2.3.GA</version>  </dependency>  <!-- Hibernate annotation -->  <dependency>  <groupId>hibernate-annotations</groupId>  <artifactId>hibernate-annotations</artifactId>  <version>3.3.0.GA</version>  </dependency>  <dependency>  <groupId>hibernate-commons-annotations</groupId>  <artifactId>hibernate-commons-annotations</artifactId>  <version>3.0.0.GA</version>  </dependency>  <!-- Hibernate library dependecy start -->  <dependency>  <groupId>dom4j</groupId>  <artifactId>dom4j</artifactId>  <version>1.6.1</version>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>commons-collections</groupId>  <artifactId>commons-collections</artifactId>  <version>3.2.1</version>  </dependency>  <dependency>  <groupId>cglib</groupId>  <artifactId>cglib</artifactId>  <version>2.2</version>  </dependency>  <!-- Hibernate library dependecy end -->  <dependency>  <groupId>javax.transaction</groupId>  <artifactId>jta</artifactId>  <version>1.1</version>  </dependency>  <dependency>  <groupId>javax.persistence</groupId>  <artifactId>persistence-api</artifactId>  <version>1.0.2</version>  </dependency>  <dependency>  <groupId>javax.persistence</groupId>  <artifactId>persistence-api</artifactId>  <version>1.0.2</version>  </dependency>  </dependencies>  ... |

1. Now build project with **mvn eclipse:eclipse** to check dependency issues with project and update the Eclipse’s project classpath
2. **Create Hibernate configuration file**

Create a Hibernate’s configuration file and put under the resources root folder, ***src/main/resources/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>  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/website</property>  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>admin</property>  <property name=*"hibernate.bytecode.use\_reflection\_optimizer"*>false</property>  <!-- JDBC connection pool settings -->  <property name=*"connection\_pool\_size"*>10</property>  <!-- Echo the SQL -->  <property name=*"show\_sql"*>true</property>  <!-- Create or Update the database schema on startup -->  <property name=*"hibernate.hbm2ddl.auto"*>update</property>    <!-- Using for xml mapping file -->  <mapping resource=*"com/website/demo/Student.hbm.xml"*></mapping>  <!-- Using for annotation -->  <mapping class=*"com.website.demo.dao.Student"*></mapping>  </session-factory>  </hibernate-configuration> |

1. Create Hibernate configuration + Model class

* **For XML Mapping**: create hibernate mapping file

Create **Student.java** class in package **src/main/java/com/website/demo/dao**

|  |
| --- |
| **package** com.website.demo.dao;  **public** **class** Student **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String firstName;  **private** String lastName;  **private** Integer age;  **public** Student() { }  **public** Student(String firstName, String lastName, Integer age) {  **this**.firstName = firstName;  **this**.lastName = lastName;  **this**.age = age;  }  // TODO: generate getter and setter method  } |

Create **Student.hbm.xml** in **src/main/resources/com/website/demo/**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.website.demo.dao.Student"* table=*"student"* catalog=*"website"*>  <id name=*"id"* type=*"long"*>  <column name=*"ID"* />  <generator class=*"identity"* />  </id>  <property name=*"firstName"* type=*"string"*>  <column name=*"FIRST\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"lastName"* type=*"string"*>  <column name=*"LAST\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"age"* type=*"integer"*>  <column name=*"AGE"* length=*"20"* not-null=*"true"*/>  </property>  </class>  </hibernate-mapping> |

* **For Annotation:** hibernate mapping on model class

Create **Student.java** class in package **src/main/java/com/website/demo/dao**

|  |
| --- |
| **package** com.website.demo.dao;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** **static** javax.persistence.GenerationType.IDENTITY;  **import** javax.persistence.Id;  **import** javax.persistence.Table;  @Entity  @Table(name = "student", catalog = "website", appliesTo = "")  **public** **class** Student **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String firstName;  **private** String lastName;  **private** Integer age;  //TODO: generate constructor for Student  @Id  @GeneratedValue(strategy = IDENTITY)  @Column(name = "ID", unique = **true**, nullable = **false**)  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }    @Column(name = "FIRST\_NAME", length = 20)  **public** String getFirstName() {  **return** firstName;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }  @Column(name = "LAST\_NAME", length = 20)  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  @Column(name = "AGE")  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  } |

1. **Create Hibernate Utility class**

To take care of Hibernate start up and retrieve the session easily

Create **HibernateUtil.java** in **src/main/java/com/website/demo/persistence**

* **For XML Mapping**

|  |
| --- |
| **package** com.website.demo.persistence;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.cfg.Configuration;  **public** **class** HibernateUtil {  **private** **static** **final** SessionFactory ***sessionFactory*** = *buildSessionFactory*();  **private** **static** SessionFactory buildSessionFactory() {  **try** {  // Create the SessionFactory from hibernate.cfg.xml  **return** **new** Configuration().configure().buildSessionFactory();  } **catch** (Throwable ex) {  // Make sure you log the exception, as it might be swallowed  System.***err***.println("Initial SessionFactory creation failed." + ex);  **throw** **new** ExceptionInInitializerError(ex);  }  }  **public** **static** SessionFactory getSessionFactory() {  **return** ***sessionFactory***;  }  **public** **static** **void** shutdown() {  // Close caches and connection pools  *getSessionFactory*().close();  }  } |

* **For Annotation Mapping**

|  |
| --- |
| **package** com.website.demo.persistence;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.cfg.AnnotationConfiguration;  **public** **class** HibernateUtil {  **private** **static** **final** SessionFactory ***sessionFactory*** = *buildSessionFactory*();  **private** **static** SessionFactory buildSessionFactory() {  **try** {  // Create the SessionFactory from hibernate.cfg.xml  **return** **new** AnnotationConfiguration().configure().buildSessionFactory();  } **catch** (Throwable ex) {  // Make sure you log the exception, as it might be swallowed  System.***err***.println("Initial SessionFactory creation failed." + ex);  **throw** **new** ExceptionInInitializerError(ex);  }  }  **public** **static** SessionFactory getSessionFactory() {  **return** ***sessionFactory***;  }  **public** **static** **void** shutdown() {  // Close caches and connection pools  *getSessionFactory*().close();  }  } |

1. **Testing**

|  |
| --- |
| **public** **static** **void** main(String[] args) {  System.***out***.println("Maven + Hibernate + MySQL");  Session session = HibernateUtil.*getSessionFactory*().openSession();  session.beginTransaction();  Student student = **new** Student();  student.setFirstName("Phuong");  student.setLastName("Mai");  student.setAge(29);  session.save(student);  session.getTransaction().commit();  } |

**Hibernate with MySQL (2)**

1. **Mapping Types**

|  |  |  |
| --- | --- | --- |
| MAPPING TYPE | JAVA TYPE | SQL TYPE |
| **integer** | *int or java.lang.Integer* | *INTEGER* |
| **long** | *long or java.lang.Long* | *BIGINT* |
| **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* |
| **binary** | *byte[]* | *VARBINARY (or BLOB)* |
| **blob** | *java.sql.Blob* | *BLOB* |
|  | | |
| **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* |

1. **Collection Mapping**

|  |  |
| --- | --- |
| COLLECTION TYPE | MAPPING |
| **<list>** | *java.util.List* |
| **<set>** | *java.util.Set* |
| **<map>** | *java.util.Map* |

1. **ORM – Object Relationship Mapping**

* **One – to – One**

**Using XML mapping**

Create **Address.java** class in package **src/main/java/com/website/demo/dao** and **Address.hbm.xml** in **src/main/resources/com/website/demo/**

|  |
| --- |
| **package** com.website.demo.dao;  **public** **class** Address **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String street;  **private** String district;  **private** String city;  //TODO: generate constructor & getter and setter methods    } |
|  |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.website.demo.dao.Address"* table=*"address"* catalog=*"website"*>  <id name=*"id"* type=*"long"*>  <column name=*"ID"* />  <generator class=*"identity"* />  </id>  <property name=*"street"* type=*"string"*>  <column name=*"STREET"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"district"* type=*"string"*>  <column name=*"DISTRICT"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"city"* type=*"string"*>  <column name=*"CITY"* length=*"20"* not-null=*"true"*/>  </property>  </class>  </hibernate-mapping> |

Update **Student.java** class in package **src/main/java/com/website/demo/dao** and **Student.hbm.xml** in **src/main/resources/com/website/demo/**

|  |
| --- |
| **package** com.website.demo.dao;  **public** **class** Student **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String firstName;  **private** String lastName;  **private** Integer age;  **private** Address address;  //TODO: generate constructor & getter and setter methods  } |
|  |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.website.demo.dao.Student"* table=*"student"* catalog=*"website"*>  <id name=*"id"* type=*"long"*>  <column name=*"STUDENT\_ID"* />  <generator class=*"identity"* />  </id>  <property name=*"firstName"* type=*"string"*>  <column name=*"FIRST\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"lastName"* type=*"string"*>  <column name=*"LAST\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"age"* type=*"integer"*>  <column name=*"AGE"* length=*"20"* not-null=*"true"*/>  </property>  <one-to-one name=*"address"* class=*"com.website.demo.dao.Address"* cascade=*"save-update"* lazy=*"false"*></one-to-one>  </class>  </hibernate-mapping> |

**Using Annotation mapping**

Create **Address.java** class in package **src/main/java/com/website/demo/dao** and Update **Student.java** class in package **src/main/java/com/website/demo/dao**

|  |
| --- |
| **package** com.website.demo.dao;  **import** **static** javax.persistence.GenerationType.***IDENTITY***;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.Id;  **import** javax.persistence.Table;  @Entity  @Table(name = "address", catalog = "website")  **public** **class** Address **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String street;  **private** String district;  **private** String city;  //TODO: generate constructor for Address  @Id  @GeneratedValue(strategy = ***IDENTITY***)  @Column(name = "ID", unique = **true**, nullable = **false**)  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }  @Column(name = "STREET", length = 20)  **public** String getStreet() {  **return** street;  }  **public** **void** setStreet(String street) {  **this**.street = street;  }  @Column(name = "DISTRICT", length = 20)  **public** String getDistrict() {  **return** district;  }  **public** **void** setDistrict(String district) {  **this**.district = district;  }  @Column(name = "CITY", length = 20)  **public** String getCity() {  **return** city;  }  **public** **void** setCity(String city) {  **this**.city = city;  }  } |
|  |
| **package** com.website.demo.dao;  **import** **static** javax.persistence.GenerationType.***IDENTITY***;  **import** java.util.List;  **import** javax.persistence.CascadeType;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.FetchType;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.Id;  **import** javax.persistence.OneToOne;  **import** javax.persistence.PrimaryKeyJoinColumn;  **import** javax.persistence.Table;  @Entity  @Table(name = "student", catalog = "website")  **public** **class** Student **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String firstName;  **private** String lastName;  **private** Integer age;  **private** Address address;  //TODO: generate constructor for Student  @Id  @GeneratedValue(strategy = ***IDENTITY***)  @Column(name = "STUDENT\_ID", unique = **true**, nullable = **false**)  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }  @Column(name = "FIRST\_NAME", length = 20)  **public** String getFirstName() {  **return** firstName;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }  @Column(name = "LAST\_NAME", length = 20)  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  @Column(name = "AGE")  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  @OneToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)  @PrimaryKeyJoinColumn  **public** Address getAddress() {  **return** address;  }  **public** **void** setAddress(Address address) {  **this**.address = address;  }  } |

* **Many – to – Many**

**Using XML Mapping**

Create **Course.java** class in package **src/main/java/com/website/demo/dao** and **Course.hbm.xml** in **src/main/resources/com/website/demo/**

|  |
| --- |
| **package** com.website.demo.dao;  **public** **class** Course **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String courseName;    //TODO: generate constructor & getter and setter methods  } |
|  |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.website.demo.dao.Course"* table=*"course"* catalog=*"website"*>  <id name=*"id"* type=*"long"*>  <column name=*"COURSE\_ID"* />  <generator class=*"identity"* />  </id>  <property name=*"courseName"* type=*"string"*>  <column name=*"COURSE\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  </class>  </hibernate-mapping> |

Update **Student.java** class in package **src/main/java/com/website/demo/dao** and **Student.hbm.xml** in **src/main/resources/com/website/demo/**

|  |
| --- |
| **package** com.website.demo.dao;  **import** java.util.List;  **public** **class** Student **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String firstName;  **private** String lastName;  **private** Integer age;  **private** Address address;  **private** List<Course> courses;  //TODO: generate constructor & getter and setter methods  } |
|  |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.website.demo.dao.Student"* table=*"student"* catalog=*"website"*>  <id name=*"id"* type=*"long"*>  <column name=*"STUDENT\_ID"* />  <generator class=*"identity"* />  </id>  <property name=*"firstName"* type=*"string"*>  <column name=*"FIRST\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"lastName"* type=*"string"*>  <column name=*"LAST\_NAME"* length=*"20"* not-null=*"true"*/>  </property>  <property name=*"age"* type=*"integer"*>  <column name=*"AGE"* length=*"20"* not-null=*"true"*/>  </property>  <one-to-one name=*"address"* class=*"com.website.demo.dao.Address"*  cascade=*"save-update"* lazy=*"false"*></one-to-one>  <list name=*"courses"* table=*"STUDENT\_COURSE"* cascade=*"all"*>  <key column=*"STUDENT\_ID"* />  <list-index column=*"index"* />  <many-to-many column=*"COURSE\_ID"* class=*"com.website.demo.dao.Course"* />  </list>  </class>  </hibernate-mapping> |

**Using Annotation Mapping**

Create **Course.java** class in package **src/main/java/com/website/demo/dao** and Update **Student.java** class in package **src/main/java/com/website/demo/dao**

|  |
| --- |
| **package** com.website.demo.dao;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** **static** javax.persistence.GenerationType.***IDENTITY***;  **import** javax.persistence.Id;  **import** javax.persistence.Table;  @Entity  @Table(name = "course", catalog = "website")  **public** **class** Course **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String courseName;  //TODO: generate constructor for Course  @Id  @GeneratedValue(strategy = ***IDENTITY***)  @Column(name = "COURSE\_ID", unique = **true**, nullable = **false**)  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }  @Column(name = "COURSE\_NAME", length = 20)  **public** String getCourseName() {  **return** courseName;  }  **public** **void** setCourseName(String courseName) {  **this**.courseName = courseName;  }  } |
|  |
| **package** com.website.demo.dao;  **import** **static** javax.persistence.GenerationType.***IDENTITY***;  **import** java.util.List;  **import** javax.persistence.CascadeType;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.FetchType;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.Id;  **import** javax.persistence.JoinTable;  **import** javax.persistence.JoinColumn;  **import** javax.persistence.ManyToMany;  **import** javax.persistence.OneToOne;  **import** javax.persistence.PrimaryKeyJoinColumn;  **import** javax.persistence.Table;  @Entity  @Table(name = "student", catalog = "website")  **public** **class** Student **implements** java.io.Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Long id;  **private** String firstName;  **private** String lastName;  **private** Integer age;  **private** Address address;  **private** List<Course> courses;  //TODO: generate constructor for Student  @Id  @GeneratedValue(strategy = ***IDENTITY***)  @Column(name = "STUDENT\_ID", unique = **true**, nullable = **false**)  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }  @Column(name = "FIRST\_NAME", length = 20)  **public** String getFirstName() {  **return** firstName;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }  @Column(name = "LAST\_NAME", length = 20)  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  @Column(name = "AGE")  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  @OneToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)  @PrimaryKeyJoinColumn  **public** Address getAddress() {  **return** address;  }  **public** **void** setAddress(Address address) {  **this**.address = address;  }  @ManyToMany(cascade = CascadeType.***ALL***)  @JoinTable(name = "STUDENT\_COURSE", joinColumns = { @JoinColumn(name = "STUDENT\_ID") }, inverseJoinColumns = { @JoinColumn(name = "COURSE\_ID") })  **public** List<Course> getCourses() {  **return** courses;  }  **public** **void** setCourses(List<Course> courses) {  **this**.courses = courses;  }  } |

* **One – to – Many / Many – to – One**

*(base on* ***one – to – one*** *and* ***many – to – many*** *for practice yourselft)   
For ex: Object* **Course** *contain* ***List<HomeWork>*** *as properties**and Object HomeWork contain* ***Course*** *as property also.*

**Hibernate with MySQL (3)**

1. **HQL – Hibernate Query Language**

the syntax is quite similar to database SQL language.

The main difference between is HQL uses **class name** instead of **table name**, and **property names** instead of **column name**.

* **SELECT**

|  |
| --- |
| Session session = HibernateUtil.*getSessionFactory*().openSession();  session.beginTransaction();  Query query = session.createQuery("from Address as A where A.city = ‘HCM’");  List list = query.list(); *// return list objects*  Query query1 = session.createQuery("select S from Student as S where S.firstName = :fName");  Query1.setParameter("fName", "Phuong");  Student student = (Student) query1.uniqueResult(); *// return single object*    session.getTransaction().commit(); |

* **UPDATE**

|  |
| --- |
| Query query = session.createQuery("update Student set firstName = :fName, lastName = :lName where id = :id");  query.setParameter("fName", "Phuong1");  query.setParameter("lName", "Mai1");  query.setParameter("id", 1);  **int** result = query.executeUpdate(); *// return number of record has been updated* |

* **DELETE**

|  |
| --- |
| Query query = session.createQuery("delete Student where id = :id");  query.setParameter("id", 1);  **int** result = query.executeUpdate(); *// return number of record has been deleted* |

* **INSERT** – HQL only support insert from another table**: INSERT INTO … SELECT ….**

|  |
| --- |
| Query query = session.createQuery("insert into Address(street, district, city) select tmp.street, tmp.district, tmp.city from Address\_temp as tmp");  **int** result = query.executeUpdate(); *// return number of record has been inserted* |

* **PAGINATION**

|  |
| --- |
| Query query = session.createQuery("from Address");  query.setFirstResult(20); *// set offset*  query.setMaxResults(10); *// set limit*  List list = query.list(); |

* **AGGREGATE** – HQL support function such as **min(…), max(…), avg(…), sum(…), count(…), etc**

|  |
| --- |
| ***// count total number of Student***  Query query = session.createQuery("select count(id) from Student");  ***// get min age of Student***  Query query = session.createQuery("select min(age) from Student ");  ***// get max number of Student***  Query query = session.createQuery("select max(age) from Student ");  ***// get average age of Student***  Query query = session.createQuery("select avg(age) from Student ");  ***// get total age of Student***  Query query = session.createQuery("select sum(age) from Student "); |

1. **HCQL – Hibernate Criteria Query Language**

|  |
| --- |
| ***// get all the records***  Criteria criteria = session.createCriteria(Student.**class**);  List students = criteria.list();  ***// pagination with limit and offset***  Criteria criteria = session.createCriteria(Student.**class**);  criteria.setFirstResult(0);  criteria.setMaxResults(10);  List students = criteria.list();  ***// set order ASC***  Criteria criteria = session.createCriteria(Student.**class**);  criteria.addOrder(Order.*asc*("age"));  List students = criteria.list();  ***// set order DES***  Criteria criteria = session.createCriteria(Student.**class**);  criteria.addOrder(Order.*desc*("age"));  List students = criteria.list();  ***// set restrictions***  Criteria criteria = session.createCriteria(Student.**class**);  criteria.add(Restrictions.*between*("age", 10, 20)); *// set between constraint*  criteria.add(Restrictions.*like*("firstName", "Phuong")); *// set like constraint*  criteria.add(Restrictions.*eq*("age", 29)); *// set equal constraint*  criteria.add(Restrictions.*ne*("age", 20)); *// set not equal constraint*  criteria.add(Restrictions.*lt*("age", 20)); *// set less than constraint*  criteria.add(Restrictions.*le*("age", 20)); *// set less than or equal constraint*  criteria.add(Restrictions.*gt*("age", 20)); *// set greater than constraint*  criteria.add(Restrictions.*ge*("age", 20)); *// set greater than or equal constraint*  List students = criteria.list(); |

1. **Native SQL** – You can use native SQL to express database queries

|  |
| --- |
| *// Entity queries*  SQLQuery query = session.createSQLQuery("SELECT \* FROM STUDENT");  query.addEntity(Student.**class**);  List students = query.list();  *// Scalar queries*  SQLQuery query = session.createSQLQuery("SELECT \* FROM STUDENT as s, ADDRESS as a where s.STUDENT\_ID = a.ID");  query.setResultTransformer(Criteria.***ALIAS\_TO\_ENTITY\_MAP***);  List students = query.list();  *// Named SQL queries*  SQLQuery query = session.createSQLQuery("SELECT \* FROM STUDENT WHERE id = :id");  query.addEntity(Student.**class**);  query.setParameter("id", 1);  List students = query.list(); |