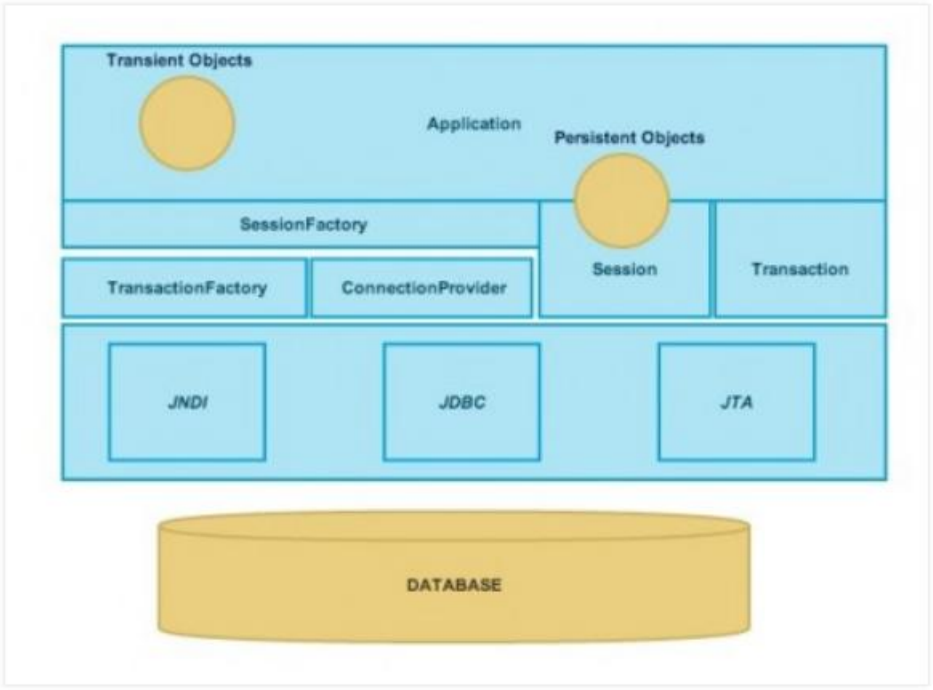
Use Of XML, Annotations and Property Configuration

Hibernate is one of the most widely used **Java ORM** tool.
Object-relational mapping or **ORM** is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.
Some of the benefits of using Hibernate as ORM tool are:

1. Hibernate supports mapping of java classes to database tables and vice versa. It provides features to perform CRUD operations across all the major relational databases.
2. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business use cases rather than making sure that database operations are not causing resource leaks.
3. Hibernate supports transaction management and make sure there is no inconsistent data present in the system.
4. Since we use XML, property files or annotations for mapping java classes to database tables, it provides an abstraction layer between application and database.
5. Hibernate helps us in mapping joins, collections, inheritance objects and we can easily visualize how our model classes are representing database tables.
6. Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.
7. Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small.
8. Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.
9. Hibernate also offers integration with some external modules. For example Hibernate Validator is the reference implementation of Bean Validation

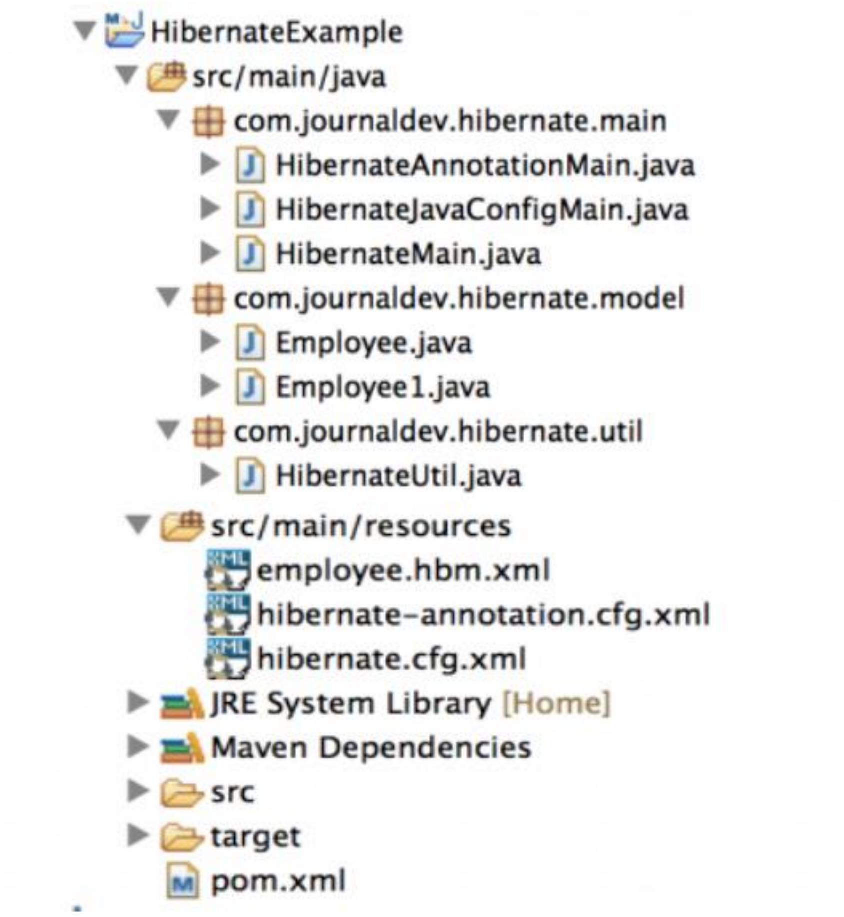
- **Hibernate Architecture**



1. SessionFactory (org.hibernate.SessionFactory)**:**
   SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We can get instance of org.hibernate.Session using SessionFactory.
2. Session (org.hibernate.Session)**:**
   Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction.
3. Persistent objects**:**
   Persistent objects are short-lived, single threaded objects that contains persistent state and business function. These can be ordinary JavaBeans/POJOs. They are associated with exactly one org.hibernate.Session.
4. Transient objects**:**
   Transient objects are persistent classes instances that are not currently associated with a org.hibernate.Session. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed org.hibernate.Session.
5. Transaction (org.hibernate.Transaction)**:**
   Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.
6. ConnectionProvider (org.hibernate.connection.ConnectionProvider)**:**
   ConnectionProvider is a factory for JDBC connections. It provides abstraction between the application and underlying javax.sql.DataSource or java.sql.DriverManager. It is not exposed to application, but it can be extended by the developer.
7. TransactionFactory (org.hibernate.TransactionFactory)**:**
   A factory for org.hibernate.Transaction instances.

- **Hibernate Example**
  When developing hibernate applications, we need to provide two set of configuration.

First set of configuration contains database specific properties that will be used to create Database connection and Session objects.
Second set of configurations contains mapping between model classes and database tables.
We can use XML based or properties based configuration for database connection related configurations. We can use XML based or annotation based configurations for providing model classes and database tables mapping.
We will use JPA annotations from javax.persistence for annotation based mappings.
Our final project will look like below image.



```
HibernateExample
  src/main/java
    com.journaldev.hibernate.main
      HibernateAnnotationMain.java
      HibernateJavaConfigMain.java
      HibernateMain.java
    com.journaldev.hibernate.model
      Employee.java
      Employee1.java
    com.journaldev.hibernate.util
      HibernateUtil.java
  src/main/resources
      employee.hbm.xml
      hibernate-annotation.cfg.xml
      hibernate.cfg.xml
  JRE System Library [Home]
  Maven Dependencies
  src
  target
  pom.xml
```

✓ **Database Table Setup**
For my example, I am using MySQL database and below script is used to create necessary

table.

```
CREATE TABLE `Employee` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `role` varchar(20) DEFAULT NULL,
  `insert_time` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=19 DEFAULT CHARSET=utf8;
```

- ✓ **Hibernate Project Dependencies**
  Our final pom.xml file looks like below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.journaldev.hibernate</groupId>
  <artifactId>HibernateExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>HibernateExample</name>

  <dependencies>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>4.3.5.Final</version>
        </dependency>
        <!-- Hibernate 4 uses Jboss logging, but older
versions slf4j for logging -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.5</version>
        </dependency>
        <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-
java</artifactId>
                <version>5.0.5</version>
        </dependency>
  </dependencies>

  <build>
        <finalName>${project.artifactId}</finalName>
  </build>
</project>
```

hibernate-core **artifact contains all the core hibernate classes, so we will get all the necessary features by including it in the project.**
Hibernate 4 uses JBoss logging but older versions uses slf4j for logging purposes, so I have included slf4j-simple artifact in my project, although not needed because I am using Hibernate 4.
mysql-connector-java **is the MySQL driver for connecting to MySQL databases, if you are using any other database then add corresponding driver artifact.**

✓ **Domain Model Classes**

As you can see in above image that we have two model classes, Employee and Employee1.
Employee is a simple Java Bean class and we will use XML based configuration for providing it's mapping details.
Employee1 is a java bean where fields are annotated with JPA annotations, so that we don't need to provide mapping in separate XML file.

```
import java.util.Date;

public class Employee {
        private int id;
        private String name;
        private String role;
        private Date insertTime;

        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getRole() {
                return role;
        }
        public void setRole(String role) {
                this.role = role;
        }
        public Date getInsertTime() {
                return insertTime;
        }
        public void setInsertTime(Date insertTime) {
                this.insertTime = insertTime;
        }

}
```

Employee class is simple java bean, there is nothing specific to discuss here.

```
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
```

```
@Entity
@Table(name="Employee",
           uniqueConstraints=
{@UniqueConstraint(columnNames={"ID"})})
public class Employee1 {

        @Id

@GeneratedValue(strategy=GenerationType.IDENTITY)
        @Column(name="ID", nullable=false,
unique=true, length=11)
        private int id;
        @Column(name="NAME", length=20, nullable=true)
        private String name;

        @Column(name="ROLE", length=20, nullable=true)
        private String role;
        @Column(name="insert_time", nullable=true)
        private Date insertTime;

        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
```

```
	public String getRole() {
		return role;
	}
	public void setRole(String role) {
		this.role = role;
	}
	public Date getInsertTime() {
		return insertTime;
	}
	public void setInsertTime(Date insertTime) {
		this.insertTime = insertTime;
	}
}
```

javax.persistence.Entity annotation is used to mark a class as Entity bean that can be persisted by hibernate, since hibernate provides JPA implementation.
javax.persistence.Table annotation is used to define the table mapping and unique constraints for the columns.
javax.persistence.Id annotation is used to define the primary key for the table.
javax.persistence.GeneratedValue is used to define that the field will be auto generated and GenerationType.IDENTITY strategy is used so that the generated "id" value is mapped to the bean and can be retrieved in the java program.
javax.persistence.Column is used to map the field with table column, we can also specify length, nullable and uniqueness for the bean properties.

- ✓ **Hibernate Mapping XML Configuration**

we will use XML based configuration for Employee class mapping. We can choose any name, but it's good to choose with table or java bean name for clarity. Our hibernate mapping file for Employee bean looks like below.
employee.hbm.xml

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

<hibernate-mapping>
        <class
name="com.journaldev.hibernate.model.Employee"
table="EMPLOYEE">
        <id name="id" type="int">
            <column name="ID" />
            <generator class="increment" />
        </id>
        <property name="name" type="java.lang.String">
            <column name="NAME" />
        </property>
        <property name="role" type="java.lang.String">
            <column name="ROLE" />
        </property>
        <property name="insertTime" type="timestamp">
                <column name="insert_time" />
        </property>
    </class>
</hibernate-mapping>
```

The xml configuration is simple and does the same thing as annotation based configuration.
Hibernate Configuration Files
We will create two hibernate configuration xml files – one for xml based configuration and another for annotation based configuration.

- hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                "-//Hibernate/Hibernate Configuration
DTD 3.0//EN"
                "http://hibernate.org/dtd/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                <!-- Database connection properties -
Driver, URL, user, password -->
                <property
name="hibernate.connection.driver_class">com.mysql.jdbc
                <property
name="hibernate.connection.url">jdbc:mysql://localhost
/TestDB</property>
                <property
name="hibernate.connection.username">pankaj</property>
                <property
name="hibernate.connection.password">pankaj123</propert
                <!-- 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</p

                <!-- 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="employee.hbm.xml" />
        </session-factory>
</hibernate-configuration>
```

Most of the properties are related to database configurations, other properties details are given in comment. Note the configuration for hibernate mapping file, we can define multiple hibernate mapping files and configure them here.

- **hibernate-annotation.cfg.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                "-//Hibernate/Hibernate Configuration
DTD 3.0//EN"
                "http://hibernate.org/dtd/hibernate-
configuration-3.0.dtd">
<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
/TestDB</property>
                <property
name="hibernate.connection.username">pankaj</property>
                <property
name="hibernate.connection.password">pankaj123</property>

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

                <!-- Mapping with model class
containing annotations -->
                <mapping
class="com.journaldev.hibernate.model.Employee1"/>
        </session-factory>
</hibernate-configuration>
```

Most of the configuration is same as XML based configuration, the only difference is the mapping configuration. We can provide mapping configuration for classes as well as packages.

- ✓ **Hibernate SessionFactory**

I have created a utility class where I am creating SessionFactory from XML based configuration as well as property based configuration. For property based configuration, we could have a property file and read it in the class, but for simplicity I am creating Properties instance in the class itself.

```
import java.util.Properties;

import org.hibernate.SessionFactory;
import
org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;
public class HibernateUtil {

        //XML based configuration
        private static SessionFactory sessionFactory;

        //Annotation based configuration
        private static SessionFactory
sessionAnnotationFactory;
    private static SessionFactory
buildSessionFactory() {
        try {
            // Create the SessionFactory from
hibernate.cfg.xml
                Configuration configuration = new
Configuration();
configuration.configure("hibernate.cfg.xml");
                System.out.println("Hibernate
Configuration loaded");

                ServiceRegistry serviceRegistry = new
                System.out.println("Hibernate
serviceRegistry created");

                SessionFactory sessionFactory =
configuration.buildSessionFactory(serviceRegistry);

            return sessionFactory;
        }
        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);
        }
    }

    private static SessionFactory
buildSessionAnnotationFactory() {
        try {
            // Create the SessionFactory from
hibernate.cfg.xml
                Configuration configuration = new
Configuration();
                configuration.configure("hibernate-
annotation.cfg.xml");
                System.out.println("Hibernate
Annotation Configuration loaded");

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

```
                ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configura
                System.out.println("Hibernate
Annotation serviceRegistry created");

                SessionFactory sessionFactory =
configuration.buildSessionFactory(serviceRegistry);

            return sessionFactory;
        }
        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);
        }
        }

    private static SessionFactory
buildSessionJavaConfigFactory() {
        try {
        Configuration configuration = new
```

```
Configuration();

                //Create Properties, can be read from
property files too
                Properties props = new Properties();

props.put("hibernate.connection.driver_class",
"com.mysql.jdbc.Driver");
                props.put("hibernate.connection.url",
"jdbc:mysql://localhost/TestDB");

props.put("hibernate.connection.username", "pankaj");
props.put("hibernate.connection.password",
"pankaj123");

props.put("hibernate.current_session_context_class",
"thread");

                configuration.setProperties(props);

                //we can set mapping file or class
with annotation
                //addClass(Employee1.class) will look
for resource
                // com/journaldev/hibernate/model
/Employee1.hbm.xml (not good)

configuration.addAnnotatedClass(Employee1.class);

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

```
        System.out.println("Hibernate Java Config
serviceRegistry created");

        SessionFactory sessionFactory =
configuration.buildSessionFactory(serviceRegistry);

        return sessionFactory;
        }
        catch (Throwable ex) {
            System.err.println("Initial SessionFactory
creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
        }

        public static SessionFactory
getSessionFactory() {
                if(sessionFactory == null)
sessionFactory = buildSessionFactory();
        return sessionFactory;
    }

        public static SessionFactory
getSessionAnnotationFactory() {
```

```
                if(sessionAnnotationFactory == null)
sessionAnnotationFactory =
buildSessionAnnotationFactory();
        return sessionAnnotationFactory;
    }

        public static SessionFactory
getSessionJavaConfigFactory() {
                if(sessionJavaConfigFactory == null)
sessionJavaConfigFactory =
buildSessionJavaConfigFactory();
        return sessionJavaConfigFactory;
    }


}
```

Creating SessionFactory for XML based configuration is same whether mapping is XML based or annotation based. For properties based, we need to set the properties in Configuration object and add annotation classes before creating the SessionFactory.
Overall creating SessionFactory includes following steps:

1. Creating Configuration object and configure it
2. Creating ServiceRegistry object and apply configuration settings.
3. Use configuration.buildSessionFactory() by passing ServiceRegistry object as argument to get the SessionFactory object.

Our application is almost ready now, let's write some test programs and execute them.

- **Hibernate XML Configuration Test**

```
import java.util.Date;

import org.hibernate.Session;

import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateMain {

        public static void main(String[] args) {
                Employee emp = new Employee();
                emp.setName("Pankaj");
                emp.setRole("CEO");
                emp.setInsertTime(new Date());

                //Get Session
                Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
                //start transaction
                session.beginTransaction();
                //Save the Model object
                session.save(emp);
                //Commit transaction
                session.getTransaction().commit();
                System.out.println("Employee
ID="+emp.getId());

                //terminate session factory, otherwise
program won't end

HibernateUtil.getSessionFactory().close();
        }

}
```

The program is self understood, when we execute the test program, we get following output.
May 06, 2014 12:40:06 AM
org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>

```
INFO: HCANN000001: Hibernate Commons Annotations
{4.0.4.Final}
May 06, 2014 12:40:06 AM org.hibernate.Version
logVersion
INFO: HHH000412: Hibernate Core {4.3.5.Final}
May 06, 2014 12:40:06 AM org.hibernate.cfg.Environment
<clinit>
INFO: HHH000206: hibernate.properties not found
May 06, 2014 12:40:06 AM org.hibernate.cfg.Environment
buildBytecodeProvider
INFO: HHH000021: Bytecode provider name : javassist
May 06, 2014 12:40:06 AM
org.hibernate.cfg.Configuration configure
INFO: HHH000043: Configuring from resource:
hibernate.cfg.xml
May 06, 2014 12:40:06 AM
org.hibernate.cfg.Configuration
getConfigurationInputStream
INFO: HHH000040: Configuration resource:
hibernate.cfg.xml
May 06, 2014 12:40:07 AM
```

```
org.hibernate.cfg.Configuration addResource
INFO: HHH000221: Reading mappings from resource:
employee.hbm.xml
May 06, 2014 12:40:08 AM
org.hibernate.cfg.Configuration doConfigure
INFO: HHH000041: Configured SessionFactory: null
Hibernate Configuration loaded
Hibernate serviceRegistry created
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure

```
WARN: HHH000402: Using Hibernate built-in connection
pool (not for production use!)
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator

```
INFO: HHH000401: using driver [com.mysql.jdbc.Driver]
at URL [jdbc:mysql://localhost/TestDB]
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator

```
INFO: HHH000046: Connection properties: {user=pankaj,
password=****}
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator

```
INFO: HHH000006: Autocommit mode: false
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure

```
INFO: HHH000115: Hibernate connection pool size: 1
(min=1)
May 06, 2014 12:40:08 AM org.hibernate.dialect.Dialect
<init>
INFO: HHH000400: Using dialect:
org.hibernate.dialect.MySQLDialect
May 06, 2014 12:40:08 AM
org.hibernate.engine.jdbc.internal.LobCreatorBuilder
useContextualLobCreation
INFO: HHH000423: Disabling contextual LOB creation as
JDBC driver reported JDBC version [3] less than 4
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService

```
INFO: HHH000399: Using default transaction strategy
(direct JDBC transactions)
May 06, 2014 12:40:08 AM
```

org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select max(ID) from EMPLOYEE
Hibernate: insert into EMPLOYEE (NAME, ROLE,
insert_time, ID) values (?, ?, ?, ?)
Employee ID=19
May 06, 2014 12:40:08 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost/TestDB]
Notice that it's printing the generated employee id, you can check database table to confirm it.

- **Hibernate Annotation Configuration Test**

```
import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.journaldev.hibernate.model.Employee1;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateAnnotationMain {

        public static void main(String[] args) {
                Employee1 emp = new Employee1();
                emp.setName("David");
                emp.setRole("Developer");
                emp.setInsertTime(new Date());

                //Get Session
                SessionFactory sessionFactory = 
HibernateUtil.getSessionAnnotationFactory();
                Session session = 
sessionFactory.getCurrentSession();
                //start transaction
                session.beginTransaction();
                //Save the Model object
                session.save(emp);
                //Commit transaction
                session.getTransaction().commit();
                System.out.println("Employee 
ID="+emp.getId());
```

```
                //terminate session factory, otherwise
program won't end
                sessionFactory.close();
        }

}
```

When we execute above program, we get following output.
May 06, 2014 12:42:22 AM
org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>

```
INFO: HCANN000001: Hibernate Commons Annotations
{4.0.4.Final}
May 06, 2014 12:42:22 AM org.hibernate.Version
logVersion
INFO: HHH000412: Hibernate Core {4.3.5.Final}
May 06, 2014 12:42:22 AM org.hibernate.cfg.Environment
<clinit>
INFO: HHH000206: hibernate.properties not found
May 06, 2014 12:42:22 AM org.hibernate.cfg.Environment
buildBytecodeProvider
INFO: HHH000021: Bytecode provider name : javassist
May 06, 2014 12:42:22 AM
org.hibernate.cfg.Configuration configure
INFO: HHH000043: Configuring from resource: hibernate-
annotation.cfg.xml
May 06, 2014 12:42:22 AM
org.hibernate.cfg.Configuration
getConfigurationInputStream
INFO: HHH000040: Configuration resource: hibernate-
annotation.cfg.xml
May 06, 2014 12:42:23 AM
org.hibernate.cfg.Configuration doConfigure
INFO: HHH000041: Configured SessionFactory: null
```

```
Hibernate Annotation Configuration loaded
Hibernate Annotation serviceRegistry created
May 06, 2014 12:42:23 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure

```
WARN: HHH000402: Using Hibernate built-in connection
pool (not for production use!)
May 06, 2014 12:42:23 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator

```
INFO: HHH000401: using driver [com.mysql.jdbc.Driver]
at URL [jdbc:mysql://localhost/TestDB]
May 06, 2014 12:42:23 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator

```
INFO: HHH000046: Connection properties: {user=pankaj,
password=****}
May 06, 2014 12:42:23 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator

```
INFO: HHH000006: Autocommit mode: false
May 06, 2014 12:42:23 AM
```

org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure

```
INFO: HHH000115: Hibernate connection pool size: 20
(min=1)
May 06, 2014 12:42:23 AM org.hibernate.dialect.Dialect
<init>
INFO: HHH000400: Using dialect:
org.hibernate.dialect.MySQL5Dialect
May 06, 2014 12:42:23 AM
org.hibernate.engine.jdbc.internal.LobCreatorBuilder
useContextualLobCreation
```

```
INFO: HHH000423: Disabling contextual LOB creation as
JDBC driver reported JDBC version [3] less than 4
May 06, 2014 12:42:23 AM
```
org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService
```
INFO: HHH000399: Using default transaction strategy
(direct JDBC transactions)
May 06, 2014 12:42:23 AM
org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory
<init>
INFO: HHH000397: Using ASTQueryTranslatorFactory
Employee ID=20
May 06, 2014 12:42:23 AM
```
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
```
INFO: HHH000030: Cleaning up connection pool
[jdbc:mysql://localhost/TestDB]
```
Have a look at the output and compare it with the output from the XML based configuration, you will notice some differences. For example, we are not setting connection pool size for annotation based configuration, so it’s setting to default value 20.

- **Hibernate Java Configuration Test**

```
import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.journaldev.hibernate.model.Employee1;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateJavaConfigMain {

        public static void main(String[] args) {
                Employee1 emp = new Employee1();
                emp.setName("Lisa");
                emp.setRole("Manager");
                emp.setInsertTime(new Date());
                //Get Session
                SessionFactory sessionFactory = 
HibernateUtil.getSessionJavaConfigFactory();
                Session session = 
sessionFactory.getCurrentSession();
                //start transaction
                session.beginTransaction();
                //Save the Model object
                session.save(emp);
                //Commit transaction
                session.getTransaction().commit();
                System.out.println("Employee 
ID="+emp.getId());
                //terminate session factory, otherwise 
program won't end
                sessionFactory.close();
        }

}
```

Output of the above test program is:

```
May 06, 2014 12:45:09 AM
org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
```

```
INFO: HCANN000001: Hibernate Commons Annotations
{4.0.4.Final}
May 06, 2014 12:45:09 AM org.hibernate.Version
logVersion
INFO: HHH000412: Hibernate Core {4.3.5.Final}
May 06, 2014 12:45:09 AM org.hibernate.cfg.Environment
<clinit>
INFO: HHH000206: hibernate.properties not found
May 06, 2014 12:45:09 AM org.hibernate.cfg.Environment
buildBytecodeProvider
INFO: HHH000021: Bytecode provider name : javassist
Hibernate Java Config serviceRegistry created
May 06, 2014 12:45:09 AM
```
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl
configure
```
WARN: HHH000402: Using Hibernate built-in connection
pool (not for production use!)
May 06, 2014 12:45:09 AM
```
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl
buildCreator
```
INFO: HHH000401: using driver [com.mysql.jdbc.Driver]
at URL [jdbc:mysql://localhost/TestDB]
May 06, 2014 12:45:09 AM
```
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl
buildCreator
```
INFO: HHH000046: Connection properties: {user=pankaj,
password=****}
May 06, 2014 12:45:09 AM
```
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl
buildCreator
INFO: HHH000006: Autocommit mode: false
May 06, 2014 12:45:09 AM
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl
configure

```
INFO: HHH000115: Hibernate connection pool size: 20
(min=1)
May 06, 2014 12:45:10 AM org.hibernate.dialect.Dialect
<init>
INFO: HHH000400: Using dialect:
org.hibernate.dialect.MySQL5Dialect
May 06, 2014 12:45:10 AM
org.hibernate.engine.jdbc.internal.LobCreatorBuilder
useContextualLobCreation
INFO: HHH000423: Disabling contextual LOB creation as
JDBC driver reported JDBC version [3] less than 4
May 06, 2014 12:45:10 AM
org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService
INFO: HHH000399: Using default transaction strategy
(direct JDBC transactions)
May 06, 2014 12:45:10 AM
org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
INFO: HHH000397: Using ASTQueryTranslatorFactory
Employee ID=21
May 06, 2014 12:45:10 AM
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool
[jdbc:mysql://localhost/TestDB]
```