1st Step

- Prepare the class to be persistent:
  - Add a surrogate id field, usually int, long, Integer, Long.
  - Encapsulate fields (properties) using exclusively getter and setter methods.
  - Make the setter of id private in order to not allow accidental update of it.
  - Add a no-parameter constructor (needed by Hibernate to use java reflection).
  - Put collection initialization online as opposed within constructors.
  - Use Set<…> as opposed to List<…> if you aren't interested in the order of elements. This because List needs additional work to get persistent and this might not be justified.

BookDescription

```
package bookstore;
import java.util.*;

public class BookDescription
{
    private int bookDescID;
    private String title;
    private String author;
    private String ISBN;
    private int price;
    private String publisher;

    private Set<Review> reviews = new HashSet<Review> ();

    public BookDescription() {} //No parameter constructor
```

BookDescription

```
public BookDescription(String title, String author, String ISBN, String
 publisher, int price)
{
   this.setTitle(title);
   this.setAuthor(author);
   this.setISBN(ISBN);
   this.setPublisher(publisher);
   this.setPrice(price);
}
…
public Set<Review> getReviews() { return reviews; }
public void setReviews(Set<Review> reviews) { this.reviews = reviews; }

public int getBookDescID() { return bookDescID; }
private void setBookDescID(int bookDescID) {
        this.bookDescID = bookDescID;
}
```

Review

```
package bookstore;

public class Review {
   private int reviewID;
   private String content;

   public Review() {}
   public Review(String content)  { this.setContent(content); }

   public String toString() { return getContent(); }

    public int getReviewID() { return reviewID; }
    private void setReviewID(int reviewID) { this.reviewID = reviewID; }

    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }
}
```

2nd Step

- Create XML mapping files in the same directory.

- Execute “Clean/Build” in order for these files to get copied each time in the right place.

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

<hibernate-mapping>
…
</>
```

BookDescription.hbm.xml

```
…
<hibernate-mapping>
   <class name="bookstore.BookDescription" table="BOOKDESCRIPTIONS">
      <id name="bookDescID" column="BOOKDESC_ID">
         <generator class="native"/>
      </id>
      <property name="title"/>
      <property name="author"/>
      <property name="ISBN"/>
      <property name="price"/>
      <property name="publisher"/>

      <set name="reviews" table="REVIEWS" lazy="false" cascade="all">
         <key column="BOOKDESC_ID"/>
         <one-to-many class="bookstore.Review"/>
      </set>
   </class>
</hibernate-mapping>
```

Review.hbm.xml

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

<hibernate-mapping>

    <class name="bookstore.Review" table="REVIEWS">
        <id name="reviewID" column="REVIEW_ID">
            <generator class="native"/>
        </id>

        <property name="content"/>
    </class>

</hibernate-mapping>
```

3rd Step - 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>

        <!-- Database connection settings -->
        <property
      name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>…
```

hibernate.cfg.xml

```
        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

        …
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>

        <mapping resource="bookstore/BookDescription.hbm.xml"/>
        <mapping resource="bookstore/Review.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

4th Step

- Add Jar files for Hibernate
  - Right Click on the project
  - Select “Properties”
  - Select “Libraries”
  - Press Add Jar/Folder

- Add Jar file for jdbc driver
  - Similar

5th Step - MyHibernate util class

```
package bookstore;
import org.hibernate.*;
import org.hibernate.cfg.*;
import java.util.*;
import java.io.*;

public class MyHibernate {
   private static final SessionFactory sessionFactory;
   static {
      try {          sessionFactory = new Configuration()
                    .configure("/bookstore/hibernate.cfg.xml")
                    .buildSessionFactory();
      } catch (Throwable ex) {
         System.err.println("Initial SessionFactory creation failed." + ex);
         throw new ExceptionInInitializerError(ex);
      }
   }
   public static SessionFactory getSessionFactory() { return sessionFactory; }
}
```

6th Step – Start Database

- Start Database HSQLDB – a free database written in Java.
- Usually for testing purposes.
- Actually, we only need the **hsqldb.jar**

- Create a “data” directory. Cd to it.
- Run HSQLDB:

java -classpath <dir>\hsqldb.jar org.hsqldb.Server

7th Step – Let's use Hibernate

```
package bookstore;
import org.hibernate.*;
import org.hibernate.criterion.*;
import java.util.*;

public class Catalog {
    private static Catalog instance = null;

    protected Catalog() {
        add("Object Oriented Analysis and Design", "Craig Larman", "11111",
                                                        "Prentice Hall", 60);
        add("A First Course in Databases", "Jeff Ullman", "22222",
                                                        "Prentice Hall", 80);
        add("Database System Implementation", "Jeff Ullman", "33333",
                                                        "Academic Press", 100);
```

7th Step – Let's use Hibernate

```
    //Let's add some reviews
    List<BookDescription> result = find(null, null, null, null);

    Session session = MyHibernate.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    for(BookDescription bd : result) {
        for(int i=1; i<=5; i++) {
            Review review = new Review("Review " + i +
                                       " for book " + bd.getTitle() );
            bd.getReviews().add(review);
        }
        session.update(bd); //Only now will the reviews be stored in DB
    }
    session.getTransaction().commit();
}
```

7th Step – Let's use Hibernate

```
public static synchronized Catalog getInstance() { … }

public void add(String title, String author, String ISBN,
                String publisher, int price) {
   Session session = MyHibernate.getSessionFactory().getCurrentSession();
   session.beginTransaction();

   BookDescription bd = new BookDescription(title, author, ISBN,
                                            publisher, price);
   session.save(bd);
   session.getTransaction().commit();
}
```

7th Step – Let's use Hibernate

```
public BookDescription find(int bookDescID) {
    BookDescription bd = null;

    Session session = MyHibernate.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Criteria criteria = session.createCriteria(BookDescription.class);
    criteria.add( Expression.eq("bookDescID", new Integer(bookDescID)) );
    List bdList = (List<BookDescription>) criteria.list();

    if (bdList != null) bd = (BookDescription)bdList.get(0);

    session.getTransaction().commit();
    return bd;
}
```

7th Step – Let's use Hibernate

```
    //null matches anything
    public List<BookDescription> find(  String title,  String author,
                                        String ISBN, String publisher) {
        List<BookDescription> result = null;
        Session session = MyHibernate.getSessionFactory().getCurrentSession();
        session.beginTransaction();

        Criteria criteria = session.createCriteria(BookDescription.class);
        if (title != null) criteria.add( Expression.eq("title", title) );
        if (author != null) criteria.add( Expression.eq("author", author) );
        if (ISBN != null) criteria.add( Expression.eq("ISBN", ISBN) );
        if (publisher != null) criteria.add( Expression.eq("publisher", publisher) );
        result = (List<BookDescription>) criteria.list();

        session.getTransaction().commit();
        return result;
    }
} //end of class
```