Lab Exercise: Inheritance Mapping and Polymorphic Queries

Task 1: As explained in the lecture on Inheritance Mapping and Polymorphic Queries, create an inheritance hierarchy with the abstract Animal class being the superclass, and the concrete Dog and Cat subclasses implementing it. And then map the inheritance relationship using the SINGLE_TABLE, JOINED and TABLE_PER_CLASS. And then persist the inheritance relationship to the database using each of the strategies.

Task 2: Once you're done with the mapping and persisting the inheritance relationship for each of the strategies, write a polymorphic query to examine their performances for polymorphism.

***The source code files for the lecture on "Inheritance Mapping and Polymorphic Queries" are available to be downloaded with this lab exercise. You could use them to complete the given tasks successfully.

You could also look at the source code inside all the downloadable files below.

Source Code: Equals and HashCode

Schema for hello-world database

```
CREATE DATABASE `hello-world`;
```

persistence.xml
(to be placed inside META-INF folder; the META-INF folder should be in the root of the classpath of your application)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
   version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

   <persistence-unit name="hello-world" transaction-type="RESOURCE_LOCAL">
      <properties>

         <!-- Database connection settings -->
         <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
         <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/hello-
world" />
         <property name="javax.persistence.jdbc.user" value="root" />
         <property name="javax.persistence.jdbc.password" value="password" />

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

         <!-- Create/update tables automatically using mapping metadata -->
         <property name="hibernate.hbm2ddl.auto" value="update" />

         <!-- Pretty print the SQL in the log file and console -->
         <property name="hibernate.format_sql" value="true" />
      </properties>

   </persistence-unit>
</persistence>
```

log4j.properties

```
# Direct to file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=hello-world.log
log4j.appender.file.MaxFileSize=2MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Direct to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Root logger option
log4j.rootLogger=OFF, stdout, file

#Log everything (this will also include the logging information configured by
"log4j.logger.org.hibernate.SQL=ALL" and
"log4j.logger.org.hibernate.type.descriptor.sql.BasicBinder=TRACE")
#log4j.logger.org.hibernate=ALL

# Show SQL statements
log4j.logger.org.hibernate.SQL=ALL

# Show the bind parameter values
log4j.logger.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

HelloWorldClient.java

```
package client;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;

import entity.Animal;
import entity.Cat;
import entity.Dog;

public class HelloWorldClient {
    public static void main(String[] args) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello-world");
        EntityManager em = emf.createEntityManager();
        EntityTransaction txn = em.getTransaction();

            try {
                txn.begin();

                //Persisting a Cat and a Dog object
                /*
                Cat cat = new Cat();
                cat.setName("Lucy");

                Dog dog = new Dog();
                dog.setName("Oliver");

                em.persist(cat);
                em.persist(dog);
                */

                //Polymorphic Query
                /*
                Query query = em.createQuery("select animal from Animal animal");
                List<Animal> animals = query.getResultList();
                for (Animal animal : animals) {
                    System.out.println(animal);
                }
                */
```

```
          //Querying Derived Class (Dog)
          /*
          List<Dog> dogs =em.createQuery("select dog from Dog dog").getResultList();
          for (Dog dog : dogs) {
          System.out.println(dog);
        }
        */

          txn.commit();
        }   catch(Exception e) {
            if(txn != null) { txn.rollback(); }
            e.printStackTrace();
        }   finally {
            if(em != null) { em.close(); }
        }

    }
}
```

Cat.java
(concrete subclass)

```
package entity;

import javax.persistence.Entity;

@Entity
public class Cat extends Animal {

   @Override
   public String makeNoise() {
      return "meow meow..";
   }

}
```

Dog.java
(concrete subclass)

```
package entity;

import javax.persistence.Entity;

@Entity
public class Dog extends Animal {

   @Override
   public String makeNoise() {
      return "woof woof..";
   }

}
```

Animal.java
(abstract superclass)

```
package entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
//@Inheritance(strategy=InheritanceType.JOINED) // to be used when you want to test
JOINED strategy for inheritance mapping
//@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS) // to be used when you want to
test TABLE_PER_CLASS (Table per concrete class) strategy for inheritance mapping
public abstract class Animal {

  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  //@GeneratedValue(strategy=GenerationType.TABLE) // to be used when using
TABLE_PER_CLASS strategy
  private Long id;

  //@Column(nullable=false) // cannot be used when using SINGLE_TABLE strategy
  private String name;

  public void setName(String name) {
            this.name = name;
  }

  public abstract String makeNoise();

  public String toString() {
    return name + " making " + makeNoise() + " noises";
  }

}
```