Softwares required

* JDK 17 or later
* IDE - Eclipse / IntelliJ
* Database - MySQL

Java - Platform independent programming

Fundamentals - Classes, Objects, Arrays, Encapsulation, Abstraction, Polymorphism, Inheritance, Exception Handling

Multithreading -

It allows an application to perform multiple tasks at the same time

ex: MS word

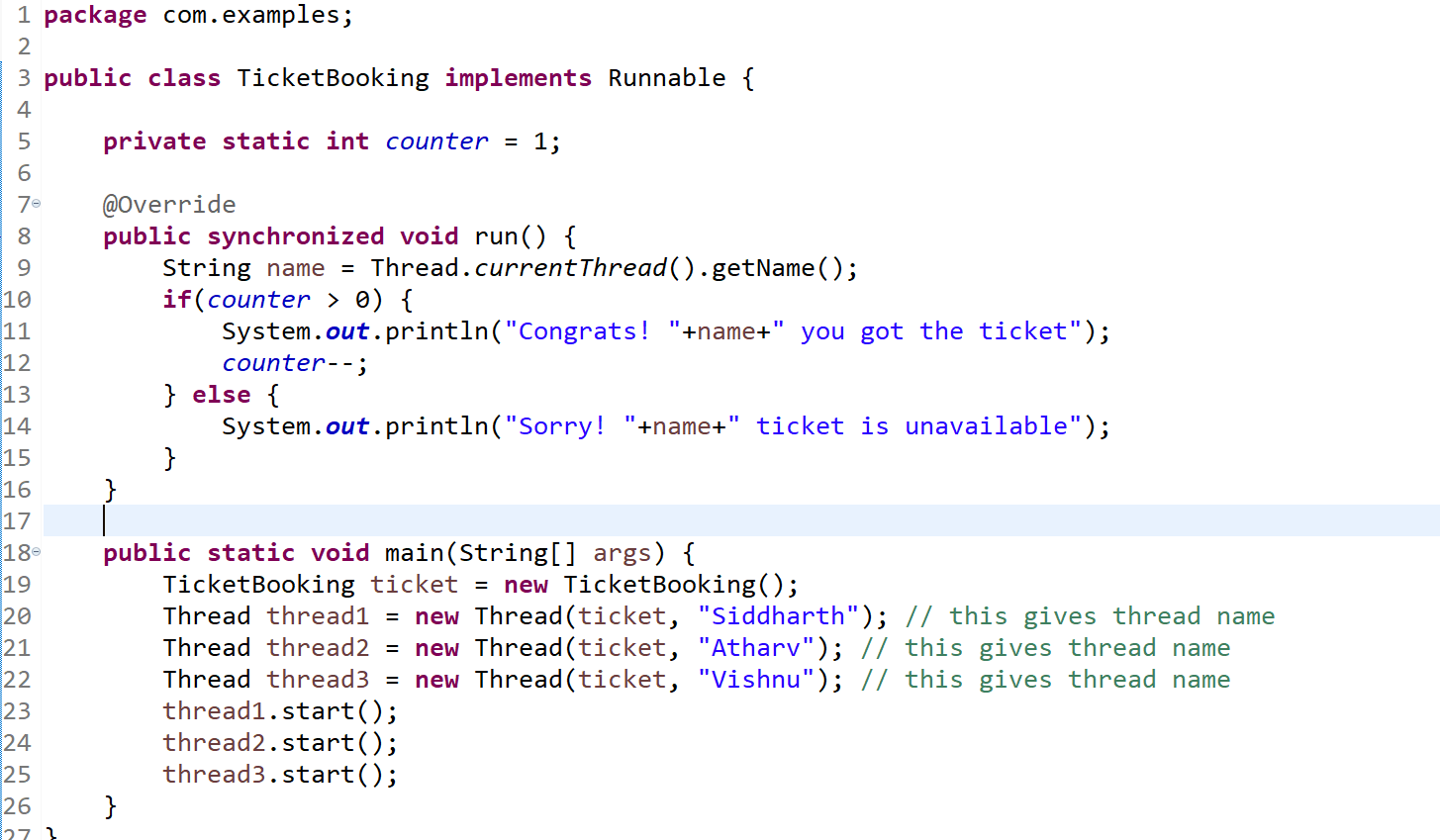
Two API’s used to create multithread programs

1. Thread class - it provides methods to manage threads - creating, starting, stopping
2. Runnable interface - it has a run method - it is an entry point for the threads

In Java application you will have a main thread which runs the main method

How the threads must work while sharing the data

* ticket booking application - tickets = 1 -> 3 users are trying to book the ticket

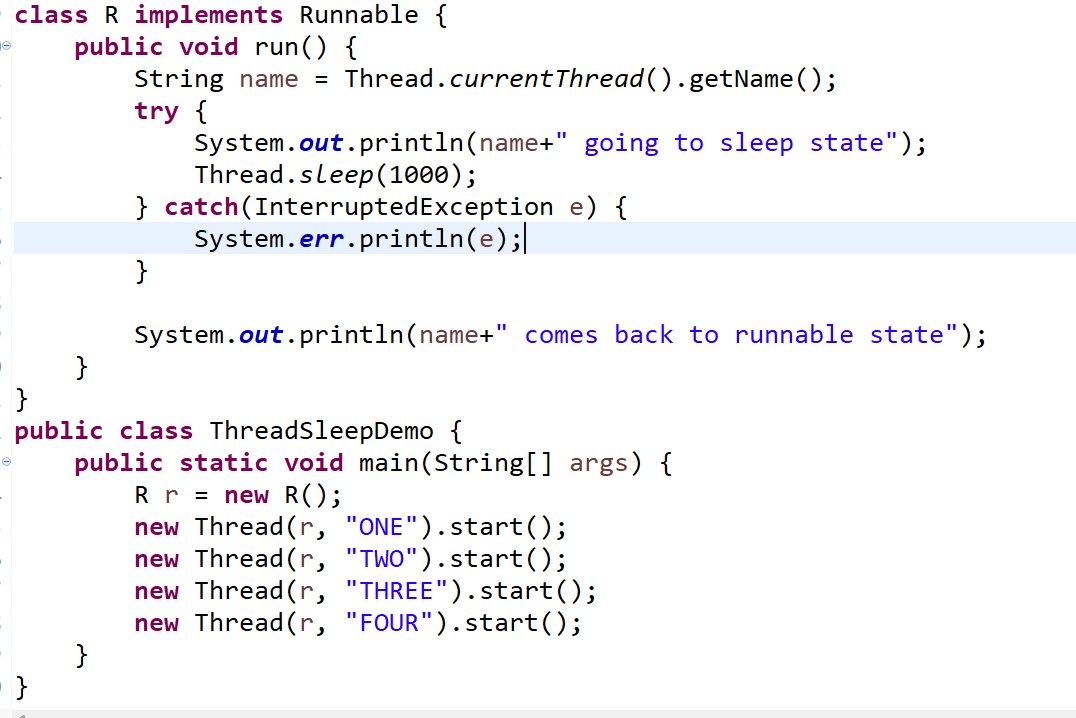


synchronized method allows only one thread to enter inside the method by acquiring the object lock, other threads should wait for the thread entered to release the lock to invoke the synchronized method

Note: You can also synchronize a particular block of code

synchronized(this) {  
 if(counter>0) { … } else {… }  
}

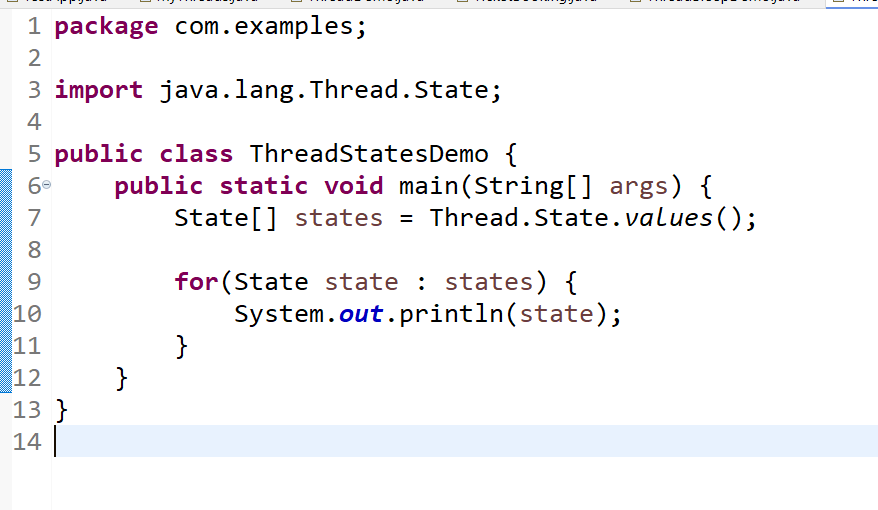
Thread.sleep(1000): Current thread goes to the sleep state allowing other idle threads to get the CPU time



Though all the threads goes to the sleep state, there will be a delay of ms in between the threads to come out of the sleep state, because once the thread goes to the sleep state, another waiting thread for the CPU time gets the CPU time & goes to the sleep state, there by with few ms all the threads will go to sleep state & come out to the runnable state after 1s delay

Thread states:

1. NEW: When you create a Thread
2. RUNNABLE: When thread gets the CPU time
3. WAITING: When thread goes to the wait state: wait() you can call to make a thread to go to wait state(), it comes out only when another thread calls notify() method, wait() & notify() are the methods present in the Object class
4. TIMED\_WAITING: When thread goes to the sleep state
5. TERMINATED: When thread is terminated



Thread pool:

It is a collection of worker threads that efficiently uses the threads, once the thread is created it will not be destroyed, it will be reused for the task

Thread t1 = new Thread();  
t1.start(); // once the thread task completes it will be destroyed, it wouldn’t be reused

In Java you have Executors & ExecutorService to use the thread pool, Executors allow you to create a thread pool of some threads which will reuse the threads for the next task without destoying, until you call the shutdown()

Design pattern

They are solutions for recurring problems

Factory pattern:

It abstracts object creation from the client which helps the client code to be loosely coupled

Developer1:  
Dao2 dao = new Dao2();  
dao.save(obj)

Developer2:  
Dao2 dao = new Dao2();  
dao.get(id);

Developer3:  
Dao2 dao = new Dao2();  
dao.get(id);

The above code is tightly coupled code, because changes done at the server side forces client side also to change, to avoid this problem we must go with loosely coupled approach, that is where factory design pattern comes

interface DBLogics {   
 save(); delete(); get(); update();  
}  
class DAO1 implements DBLogics { …. }  
class DAO2 implements DBLogics { …. }

Developer1  
Developer2  
Deverloper3

DBLogics dao = new DAO1();// still this is tightly coupled  
DBLogics dao = FactoryPattern.getDao(); // loosely coupled  
Developer who writes the factory pattern

static DBLogics getDao() {   
 return new DAO2(); //   
}

Builder pattern

It is a pattern to create a complex object step by step when you have many optional properties to be initialized and creating constructors for such properties will be cumbersome

Best example is we are going to discuss this object in Spring boot

an object called response must have following data like header, status, body and so on

ResponseEntity.body(“some content”): Only content will be sent to the client

ResponseEntity.status(200).body(“some content”): content & status both will be sent to the client

ResponseEntity.cookie(…).status(200).body(“some content”).

Observer pattern:

To define one to many relationship between objects, when one object changes its state, all dependent objects(observers) get automatically notified and updated.

object(publish a message) & multiple objects(subscribers) are connected to the publisher, then once the publisher publishes a message all the subscribers receive that message.

Ex: Youtube channel -> a new video is updated then all the subscribers are notified

Ex: Telecom services -> publish some offers then all the subscribers would receive the offers

JMS models - Java Messaging Service

Singleton pattern: It creates only one object of the class no matter how many times you request

Ex: Task manager

Activity:

Modify existing DBLogics & DAO program so that the factory pattern returns only one object when you call getDao() method multiple times, create a constructor and print the statement in the constructor when the object is created, so that when getDao() is invoked multiple times you shouldn’t see constructor print statements more than once

Java 8 or later features

* New Date & Time APIs
* Changes in the interface - default methods, static methods
* Functional interface -> you can pass function as an argument instead of an object -> functional interfaces will have only one abstract method
* Stream APIs
* JShell
* Record
* New HttpClients to work with Http2 protocol
* Sealed classes

Functional interface: It supports passing functions as a parameter instead of passing an object, it simplifies writing implementations to the interfaces with only one abstract method

The changes done to the interface in Java 8

1. Default methods can be included in interface which can be overridden in the subclass else the default implementation is considered
2. Static methods can be included in interface

Java 8 Streams:

These are sequence of elements that can process huge data in a collection without altering it, it provides many methods that takes functional interface as a parameter to perform complex operations in a simple way like filtering, sorting, collecting, mapping and so on.

Requirement:   
A complex type Dish needs to processed like you want to have list of Dish that are veg type and having 4 star rating ranging in ascending order price

List<Dish> list = new ArrayList<>();  
list.add(…); // 100’s of dish

List<Dish> vegDish = new ArrayList<>();  
iterate the list and write a if condition to filter veg and add into vegDish

you will also write if condition with 4 star rating

you will sort the dish in ascending order price

Streams will simplify these operations that can be done in a single line, because most of the methods are self iterative and there are lot of inbuilt methods to perform various operations like sorting, filtering, mapping, collecting.

for(…) {   
 set.add(“…”)  
}

Day 2:

1. Servlets/JSP
2. Hibernate

Servlets:

They are used to create dynamic web pages by running java programs at the server side

To create Servlet you must implement one of the inbuilt interface called Servlet interface that is provided by Java, this interface has 5 methods

1. init(): It is executed when servlet object is created
2. service(): It is executed whenever user sends the request & sends response, it must’ve request processing logic
3. destroy(): It is executed when servlet object is removed from the server memory, you can have clean up operations in it
4. getServletConfig(): It gives servlet configurations done in a configuration file
5. getServletInfo(): It gives servlet description

Life cycle methods: These are executed automatically by the server at the time of object creation, request processing & deletion, init(), service() & destroy() are the 3 life cycle methods in Servlet.

Java provided another servlet to reduce the load on the developers by implementing Servlet interface methods except service method, that servlet is called GenericServlet, as a developer you need to extend GenericServlet & provide logics in service method

Things you write inside service method

1. Handling protocols - http, smtp, ftp
2. Processing requests
3. Generating response

Java has provided HttpServlet that extends GenericServlet, HttpServlet handles http protocol in the service method and provides empty methods like doGet(), doPost(), doPut(), doDelete() which you can override in your class by extending HttpServlet

Project - Dynamic Web Project - Server Runtime Environment -> Create Servlets, JSP, HTML, JS, CSS

JSP: It is also used to create dynamic web pages, but it uses HTML kind of coding

Servlet

String name = “Kishor”;

out.println(“<h2>Name = “+name+”</h2>”);

JSP

<h2>${name}</h2>

Hibernate

It is a ORM framework, which maps the Java object to the database table, it takes care of lot of internal operations like converting java type to sql type or generating sql queries, creating connection pool, it does all these with the help inbuilt classes & interfaces

Hibernate uses a configuration file called hibernate.cfg.xml(you can have different name, but this is the default name), it will have database informations like url, username, password & driver class.

hibernate.cfg.xml file must be present in your classpath, if its maven & java project you must keep in src/main/resources folder, if its maven & web project you must keep in WEB-INF folder

Note: hibernate.cfg.xml is a one time configuration for the database

Hibernate uses entity class that will have the table informations

Entity class will have table & column information’s so that hibernate can take care of mapping java objects with the table as per the annotations you have in the entity

Note: In hibernate every entity must have a primary key, because hibernate performs various operations by generating the query using the primary key

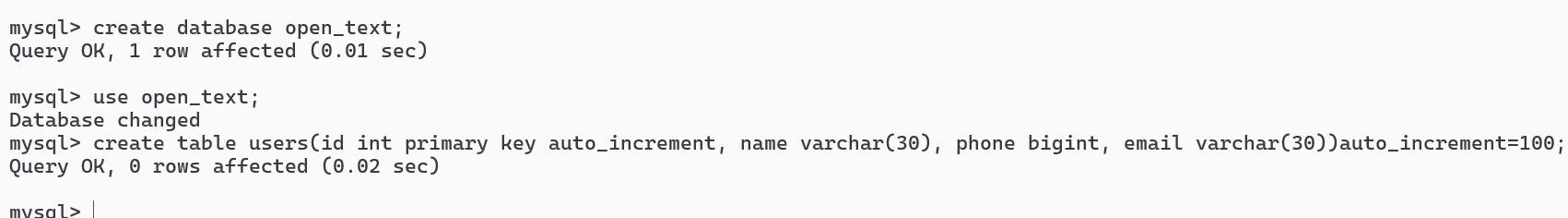
Setup required

1. Maven project
2. hibernate-core dependency : it has hibernate api’s
3. mysql-connector or postgres dependency: it is to provide the driver class
4. hibernate.cfg.xml: database & entity class information
5. Entity class: maps to the table
6. DBLogics or Client program to perform CRUD operations

Steps

1. Users table with id, name, phone & email columns
2. Java project -> Maven project -> pom.xml - add dependencies (hibernate, mysql|postgres)
3. hibernate.cfg.xml -> Configure
4. Entity class -> Users table
5. Client program

Step1:



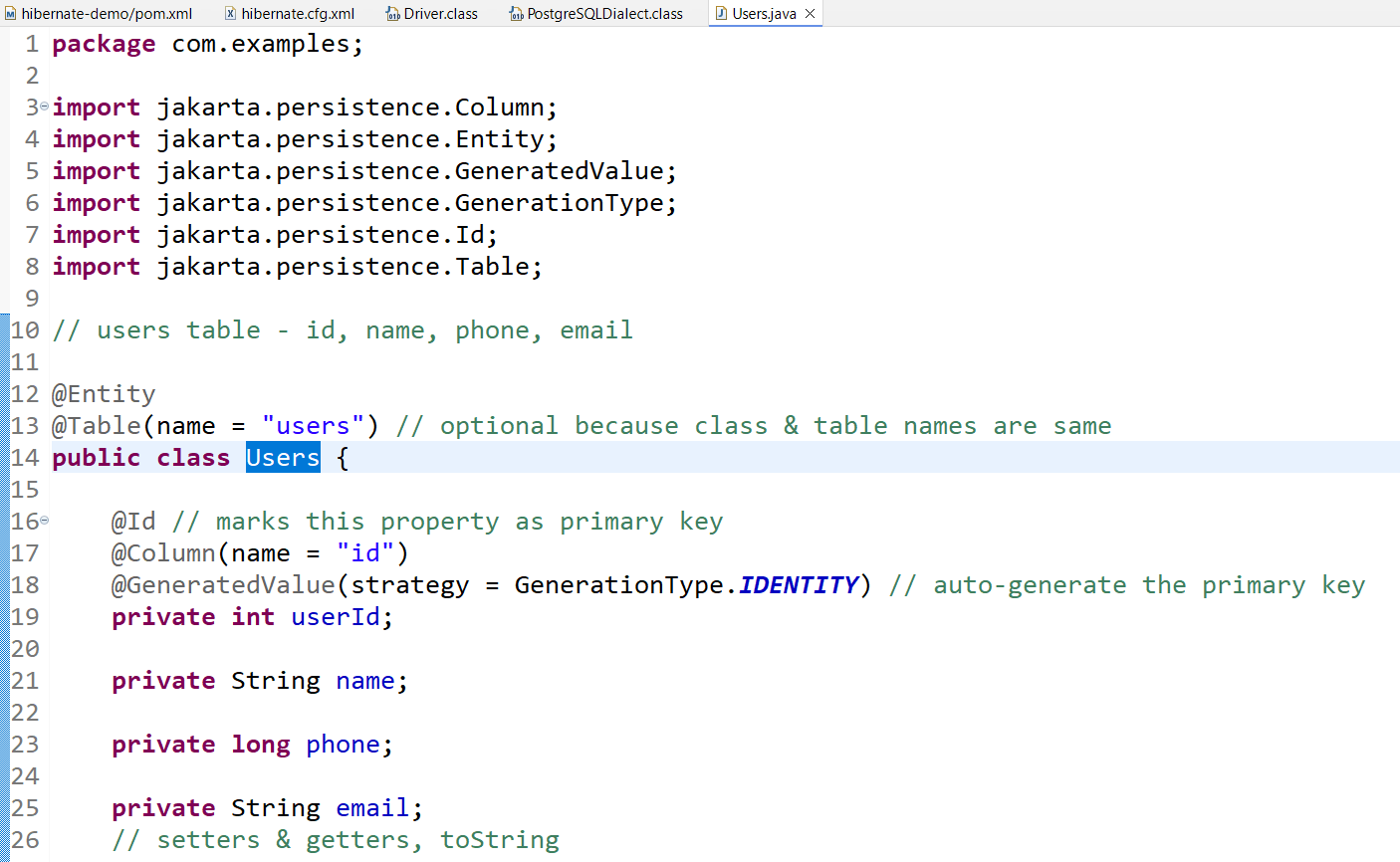
Step3

hibernate.cfg.xml



Step4:

Users.java



Step5:

Using hibernate api’s to perform operations like insert, update, delete & retrieve

These are list of API’s you use

1. Configuration: to load the hibernate.cfg.xml
2. SessionFactory: create a connection factory using the Configuration
3. Session: create a connection to the database and provide methods to perform crud operations
4. Transaction: created from the Session to perform insert, update & delete operation, only on commit() the data will be saved you can also use rollback() to undo the changes

HQL: Hibernate Query Language

It is used to write queries for entity instead of writing to the table, it is database independent

SQL: select column(s) from table\_name where column=value;

HQL: select u.name, u.phone, u.userId, u.email from Users u where u.userId=?1;

Criteria: It is used to programmatically query the entity, it is an alternative to the HQL

CriteriaBuilder : an instance for CriteriaQuery, which provides various operations like between, equals, ordering and so on

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery: this represents what result you are expecting

CriteriaQuery<T> criteria = builder.createQuery(T.class); // String, Users, Integer

Next you need to specify on which entity you need to query

Root<Users> root = criteria.from(Users.class); // query the Users entity

criteria.select(root.get(“name”));

criteria.where(builder.between(…));

session.createQuery(criteria).getResultList();

Association mapping:

When an entity is dependent on another entity then you can have this association mapping

ex: Customer & Account are interdependent, States & Cities are interdependent, Profile & Contact are interdependent, Student & Course are also interdependent

There are totally 4 types of mappings

1. one to one: customer & account
2. one to many: states & cities
3. many to one: states & cities
4. many to many: student & course

One to One

Customer

|  |  |  |
| --- | --- | --- |
| customer\_id(PK) | customer\_name | customer\_phone |
| 100 | A | 9393 |
| 200 | B | 444 |
| 300 | C | 493939 |
|  |  |  |

Account

|  |  |  |
| --- | --- | --- |
| account\_no | balance | customer\_id\_ref (FK) |
| 12345 | 5000 | 100 |
| 56789 | 6000 | 200 |
| 87634 | 2000 | 300 |

One to many or many to one

States & Cities

State

|  |  |
| --- | --- |
| state\_id (PK) | state\_name |
| KA | karanataka |
| MH | maharastra |
| TN | tamilnadu |

City

|  |  |  |
| --- | --- | --- |
| city\_id (PK) | city\_name | state\_id\_ref(FK) |
| BLR | Bangalore | KA |
| MYS | Mysore | KA |
| CHN | Chennai | TN |
| PUN | Pune | MH |
| MBI | Mumbai | MH |
| CBR | Coimbotore | TN |

Note: You can have the mappings between state & city in the 3rd table

City ->M-0-> State->0-M->City

Many to Many

Student & Course

Student

|  |  |
| --- | --- |
| roll\_no | name |
| 1 | A |
| 2 | B |
| 3 | C |

Course

|  |  |
| --- | --- |
| course\_no | course\_name |
| 10 | Java |
| 20 | C# |
| 30 | Python |
| 40 | React |

Student\_Course\_REF

|  |  |
| --- | --- |
| student\_ref (FK) | course\_ref (FK) |
| 1 | 10 |
| 1 | 20 |
| 2 | 40 |
| 3 | 40 |
| 3 | 20 |

In Hibernate we have annotations to join the tables & add the result to the java object or collection

1. @OneToOne
2. @OneToMany
3. @ManyToOne
4. @ManyToMany

@Entity  
@Table(name = “state”)  
class State {   
 String stateId;  
 String stateName;  
 @OneToMany  
 @JoinTable(name=”state\_city\_ref”, joinColumns=”state\_ref”, inverseJoinColumns=”city\_ref”)  
 List<City> cities;  
}  
  
@Entity  
@Table(name = “city”)  
class City {   
 String cityId;   
 String cityName;  
}