Oracle Technology Network
Developer Day

ORACLE
TECHNOLOGY NETWORK

OTN Developer Day: Oracle Big Data

Hands On Lab Manual

Introduction to Oracle NoSQL Database

ORACLE NOSQL DATABASE HANDS-ON WORKSHOP

Oracle NoSQL Database

Lab Exercise 1 – Start and run the Movieplex application.

In this lab, you will open and run the MovieDemo application using JDeveloper Studio.

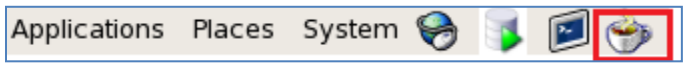
1. Open a terminal window
2. There are three Oracle NoSQL Database specific environment variables. KVHOME is where binaries are installed, KVROOT is where data files and config files are saved and KVDEMOHOME is where source of hands-on-lab project is saved.

```
echo  $KVROOT
echo $KVHOME
echo $KVDEMOHOME
```

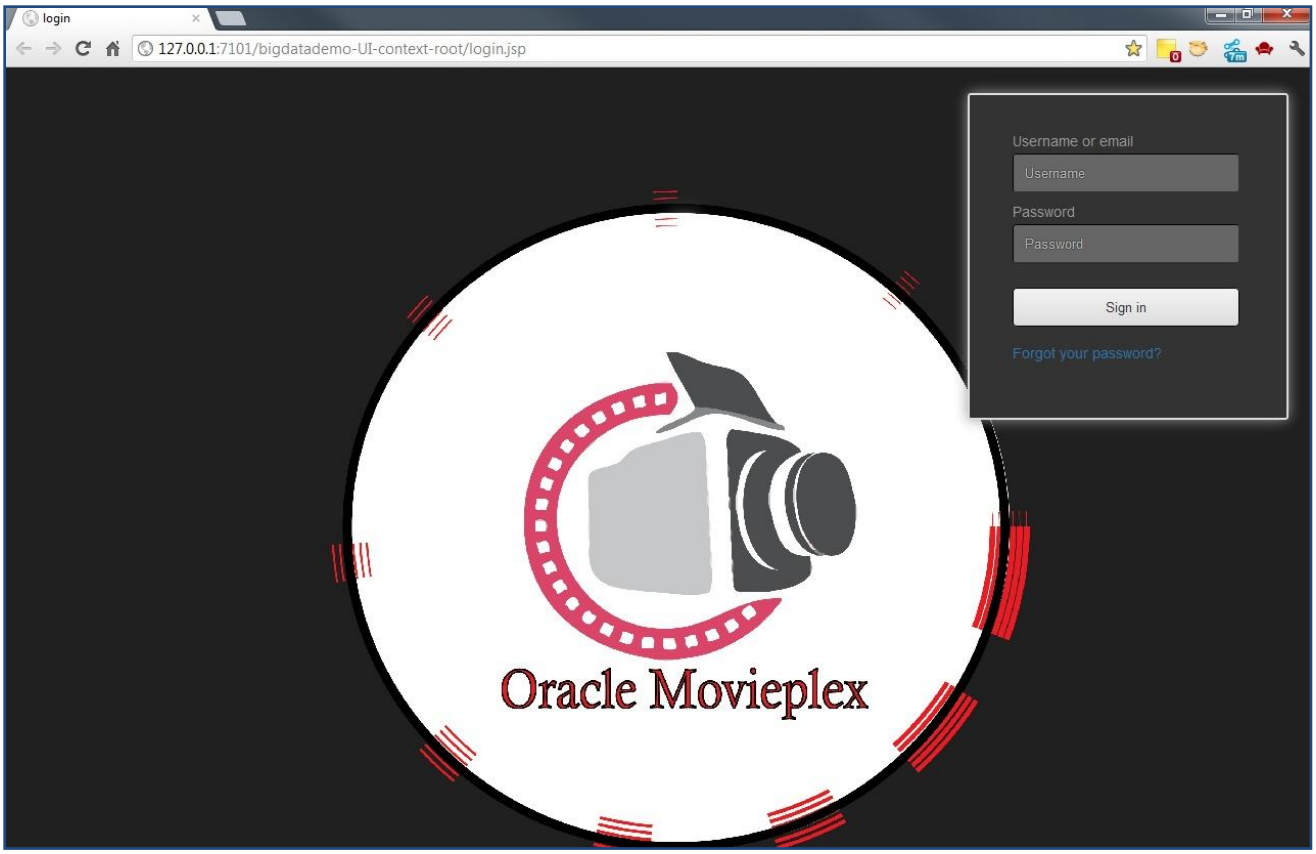
3. Make sure $KVROOT does not exist already

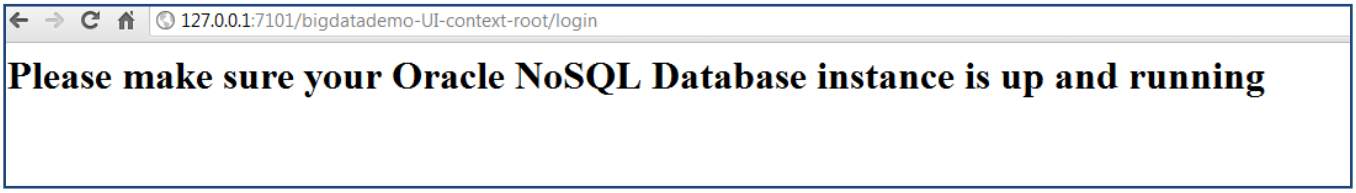
```
rm  -rf  $KVROOT
```

4. Open JDeveloper Studio by clicking the 'cup' icon from the toolbar.



5. If the `bigdatademo` project is not open inside JDeveloper Studio, open `bigdatademo.jws` from `/home/oracle/movie/moviedemo/nosqldb/bigdatademo` directory `($KVDEMOHOME)`.
6. In the Application Navigator window, expand the UI project and locate the `index.jsp` file under `Web Content` directory.
7. Right mouse click `index.jsp` and select **Run** from the context menu. JDeveloper builds and deploys the application.
8. When you see the web page below, enter the username and password (`guest1/welcome1`) and click **Sign in**.
9. What is the result?





The error message above appears because we have not started an instance of Oracle NoSQL Database. In addition, we have not loaded the profile data that contains the guest accounts. In lab exercise 2, we will start an instance of Oracle NoSQL Database and load the profile data.

Lab Exercise 2 – Start Oracle NoSQL Database instance and load the user profile data

In this exercise, you will start an Oracle NoSQL Database instance and then load the user profile information. You will step through some of the code while loading the profile information. At the end of this lab, you should be able to log on successfully. KVLite will be used as the Oracle NoSQL Database Instance. A very brief introduction to KVLite follows:

Single Node Installation – Introducing KVLite

- Run Oracle NoSQL Database on a single machine.
- Develop and test access to Oracle NoSQL Database.
- Easy to get started.
- We will use KVLite to run and debug the Movieplex application.
- KVLite is not intended for production.
- Runs in a single process.

User profile data that is going to be loaded into Oracle NoSQL Database as JSON objects and the file that contains the profile information can be found here: `$KVDEMOHOME/dataparser/metadata/customer.out`

```
{"id":1,"name":"Frankie","email":"Frankie.Nash@oraclemail.com","username":"guest1
","password":"µn\u000bN¤-\"ƒ¾çbR\\-I\u000f"}
{"id":2,"name":"Kelli","email":"Kelli.Miller@oraclemail.com","username":"guest2",
"password":"µn\u000bN¤-\"ƒ¾çbR\\-I\u000f"}
…
```

In Oracle NoSQL Database, customer profile information is going to be represented as following key-value structure:

Major-Key[1]	Major-Key[2] (username)	Minor-Key[1]	Value
CUST	guest1	info	`{"id":1,"name":"Frankie","email":"Frankie.Nash@oraclemail.com","username":"guest1","password":"µn\u000bN¤-\"ƒ¾çbR\\-I\u000f"}`
CUST	guest2	info	`{"id":2,"name":"Kelli","email":"Kelli.Miller@oraclemail.com","username":"guest2","password":"µn\u000bN¤-\"ƒ¾çbR\\-I\u000f"}`

One can store these JSON objects as plain byte arrays (same as in R1) but for efficient storage utilization and ease of use, we are going to use Avro JSON binding to serialize and deserialize user profile information to/from the NoSQL database. There are couples of benefits of using Avro SerDe APIs provided in R2 such as:

- Avro uses schema definition to parse the JSON objects and in the process removes the field's names from the object. Making it very efficient to store and retrieve from the storage layer.
- Avro support schema evolution, which means you can add columns as the schema evolves.
- Different bindings available in the product make serialization/deserialization easy for the end user. We are using JSON binding in our example code.
- Most importantly in future releases secondary index etc would leverage the schema definition.

Let's now create an Avro schema (represented as JSON only) and we will use '*ddl add-schema*' admin command to register it to the store. All the schemas definition files are available under `$KVDEMOHOME/dataparser/schemas` directory and the one that we are going to use for user profile is `customer.avsc`. The content of the file looks like this:

```
{
  "type" : "record",
  "name" : "Customer",
  "namespace" : "oracle.avro",
  "fields" : [ {
    "name" : "id",
    "type" : "int",
    "default" : 0
  }, {
    "name" : "name",
    "type" : "string",
    "default" : ""
  }, {
    "name" : "email",
    "type" : "string",
    "default" : ""
  }, {
    "name" : "username",
    "type" : "string",
    "default" : ""
  }, {
    "name" : "password",
    "type" : "string",
    "default" : ""
  } ]
}
```

Instructions

Bring up a command prompt (terminal window) and go to your `KVHOME` directory

1. Start KVLite from the `current working` directory:

```
java -jar $KVHOME/lib/kvstore.jar kvlite -host localhost -root $KVROOT
```

2. Look for a response similar to what is listed below. Minimize window (leave running).

```
java -jar $KVHOME/lib/kvstore-2.*.jar kvlite -root  $KVROOT
Created new kvlite store with args:
-root /u02/kvroot -store kvstore -host localhost -port 5000 -
admin 5001
```

3. Open a new tab in the terminal window and change directory to *schemas* directory.

```
cd $KVDEMOHOME/dataparser/schemas
```

4. Under the schemas directory you would find six AVRO schema files (*.avsc). You can cat them one by one to see how the schema is defined.

```
ls -altr
```

5. Now from the same terminal window start an admin session:

```
java -jar $KVHOME/lib/kvstore.jar runadmin -host localhost -port 5000
```

You should be logged in KV shell:

```
kv->
```

6. We are now going to register customer schema `$KVDEMOHOME/dataparser/schemas/customer.avsc`

```
Kv-> ddl add-schema -file customer.avsc
```

This is what you should get after you successfully registering the Customer schema:

```
Added schema: oracle.avro.Customer.1
kv->
```

7. Next register movie schema `$KVDEMOHOME/dataparser/schemas/movie.avsc`

```
Kv-> ddl add-schema -file movie.avsc
```

This is what you should get after you successfully registering the schema:

```
Added schema: oracle.avro.Movie.2
kv->
```

8. We need to register few more schemas as mentioned earlier:

```
Kv-> ddl add-schema -file cast.avsc
```

```
Kv-> ddl add-schema -file crew.avsc
```

```
Kv-> ddl add-schema -file genre.avsc
```

```
Kv-> ddl add-schema -file activity.avsc
```

9. Run '*show schemas*' command to make sure all six schemas are registered.

```
Kv-> show schemas
oracle.avro.Customer
  ID: 1  Modified: 2013-01-29 05:38:41 UTC, From: localhost
oracle.avro.Movie
  ID: 2  Modified: 2013-01-29 05:39:53 UTC, From: localhost
oracle.avro.Cast
  ID: 3  Modified: 2013-01-29 05:39:30 UTC, From: localhost
oracle.avro.Crew
  ID: 4  Modified: 2013-01-29 05:39:39 UTC, From: localhost
oracle.avro.Genre
  ID: 5  Modified: 2013-01-29 05:39:45 UTC, From: localhost
oracle.avro.Activity
  ID: 6  Modified: 2013-01-29 05:39:53 UTC, From: localhost
```

10. In JDeveloper Studio, expand `dataparser` project and expand all the packages by clicking '*Application Sources*'. Look for the package `oracle.demo.oow.bd.dao`, this is where Data Access Classes are defined, which means classes that interact with the Oracle NoSQL Database. Open `oracle.demo.oow.bd.dao.CustomerDAO`. This class has all the access methods for Customer related operation like creating a customer profile, reading a profile using customerId, authenticating a customer based on username and password. Let's closely examine how Customer data is written and accessed from this class. Use **CTRL + G** to go to any line number:

    - Line 55: The constructor where we create `JsonAvroBinding` and parse customer schema. This binding we later use while serializing and deserializing the customer JSON object.
    - Line 467: Method `toValue(CustomerTO)` takes `CutomerTO` POJO object and serializes into Value object
    - Line 489: Method `getCustomerTO(Value)` de-serializes Value object back into `CustomerTO` POJO.

11. Couple more things: In the same `CustomerDAO` class:

    - Line 274: Method `insertCustomerProfile(CustomerTO, boolean)` constructs a unique key and uses method `toValue(CustomerTO)` to store the customer profile into database.

```
280         if (customerTO != null) {
281             // key=/CUST/userName
282             key = KeyUtil.getCustomerKey(customerTO.getUserName());
283             // serialize CustomerTO to byte array using JSONAvroBinding
284             value = this.toValue(customerTO);
```

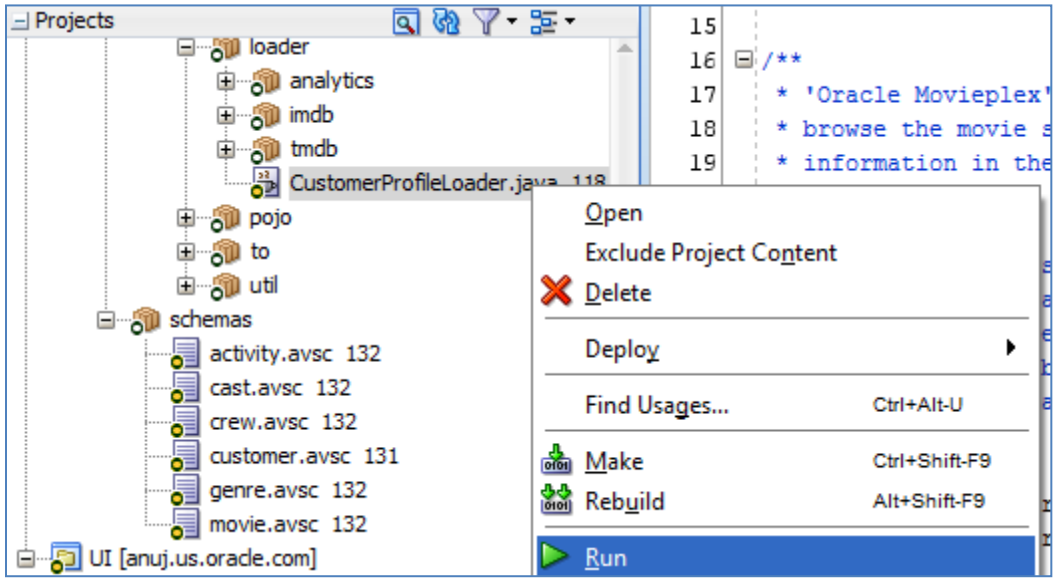
    - Line 320: Method `getCustomerByCredential(username, password)` validates customer credentials and return the profile information back as `CustomerTO` POJO object. To deserialize the data back from database we uses the method `getCustomerTO(Value)` that we defined earlier.

```
329             key = KeyUtil.getCustomerKey(username);
330             vv = getKVStore().get(key);
331
332             if (vv != null) {
333                 //deserialize the value object
334                 customerTO = this.getCustomerTO(vv.getValue());
```

12. Let's upload the customer profile data into NoSQL Database now. Open java class `oracle.demo.oow.bd.loader.CustomerProfileLoaderFile`

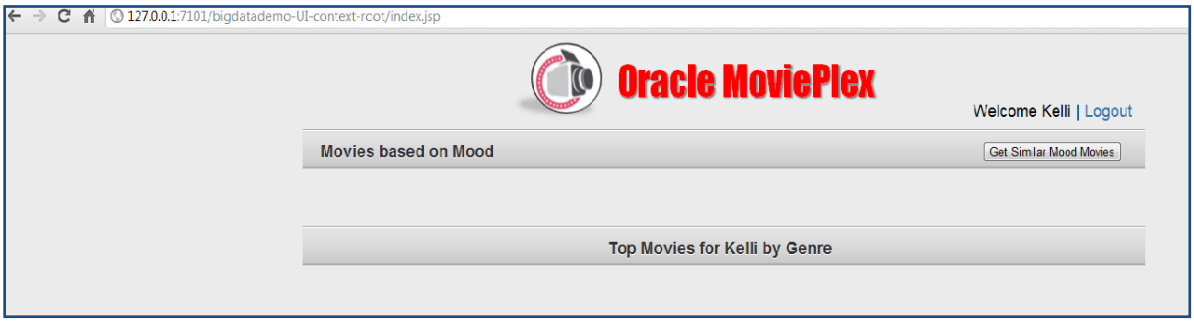
    - This class open the file `$KVDEMOHOME/dataparser/metadata/customer.out` and reads one row at a time.
    - It then parses the JSON row into `CustomerTO` POJO object and calls `insertCustomerProfile(CustomerTO, boolean)` to insert customer profile information.

13. To run, right click the class from the left pane and select 'Run', which would upload all the profile information (100 of them) into the NoSQL database.



14. To confirm, login from the Movieplex web page again. You should see a welcome page as shown in the image below.

15. Notice that there are no movies available yet. Lab exercise 3 will load the movies.



Lab Exercise 3 – Load Movie Data

In this exercise, you will load the movie data, and will create movie to genre association using major-minor keys.

Movie information data is represented as JSON structure and details of quarter million movie tiles can be found from file `$KVDEMOHOME/dataparser/metadata/wikipedia/movie-info.out`


```
{"id":857,"original title":"Saving Private
Ryan","release_date":"1998","overview":"On 6 June 1944, members of the 2nd Ranger
Battalion under Cpt. Miller fight ashore to secure a
beachhead…","vote_count":394695,"popularity":8.5,"poster_path":"/9UwfRlvq6Eekewf3
QAW5Plx0iEL.jpg","runtime":0,"genres":[{"id":"3","name":"Drama"},{"id":"18","name
":"War"},{"id":"7","name":"Action"},{"id":"1","name":"History"}]}
…
```


To search movies by movieID, movie JSON is stored in Oracle NoSQL Database, as following key-value structure:

Major-Key[1]	Major-Key[2] (MovieID)	Value
MV	829	{"id":829,"original_title":"Chinatown","release_date":"1974","overview":"JJ 'Jake' Gittes is a private detective who seems to specialize in matrimonial cases...", "vote_count":110050,"popularity":8.4,"poster_path":"/6ybT8RbSbd4AltIDABuv39dgqMU.jpg","runtime":0,**"genres":[{"id":"8","name":"Crime"},{"id":"3","name":"Drama"},{"id":"20","name":"Mystery"},{"id":"9","name":"Thriller"}]**}
MV	553	{"id":553,"original_title":"Dogville","release_date":"2003","overview":"Late one night ..", "vote_count":63536,"popularity":8,"poster_path":"/ydEUCkbkdCizibt7BQehQ5cGROO.jpg","runtime":0,**"genres":[{"id":"3","name":"Drama"},{"id":"20","name":"Mystery"},{"id":"9","name":"Thriller"}]**}

Movie schema is created to match the movie JSON string, and JSON binding is later used in the code to serialize and deserialize the JSON object to/from the KV Store. To view the movie schema you can double click *movie.avsc* from JDEV. There are few more schemas that we are going to use to store Genre, Cast, Crew, & user Activity information into the store. Feel free to view the definition by clicking the *.avsc* files.

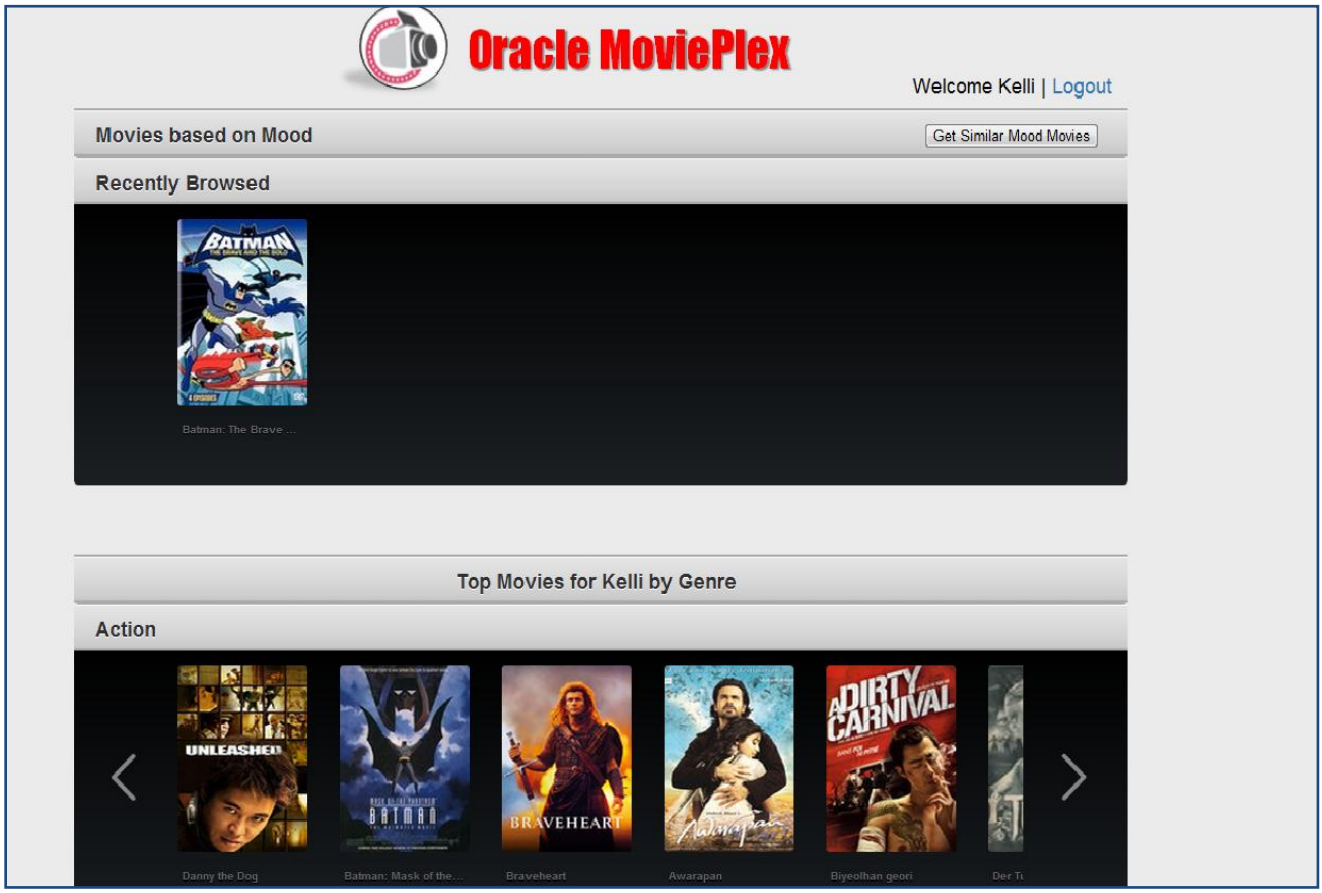
To search movies by genreID, an association between genreID and movieID(s) is created in the store. Please note that only major-minor Key associations are defined with 'Value' set to an empty string.

Major-Key[1]	Major-Key[2] (GenreID)	Minor-Key[1] (MovieID)	Value
GN_MV	8	829	""
GN_MV	3	829	""
GN_MV	20	829	""
GN_MV	9	829	""
GN_MV	3	553	""
GN_MV	20	553	""
GN_MV	9	553	""

Instructions:

1. Log out of the Movieplex application.
2. In JDeveloper Studio open `oracle.demo.oow.bd.loader.wikipwdia.`**`MovieUploader.java`**. This class reads the `movie-info.out` file and loads the content into Oracle NoSQL DB.
3. In `main()`, you will find `uploadMovieInfo()` method is called (under step 1) which is going to write information of 5000 movies (by default) .
4. Run (no need to debug) `MovieUploader` from the JDev.
5. [**Optional**] If you want to learn how movie JSON data is stored into Oracle NoSQL DB then open `oracle.demo.oow.bd.dao.MovieDAO.java` into JDev and step through the method `insertMovie(movieTO)`
6. Sign in again using the same guest id and password you used earlier.
7. Click a movie image to read its description.

8. Close the window and observe what happens.

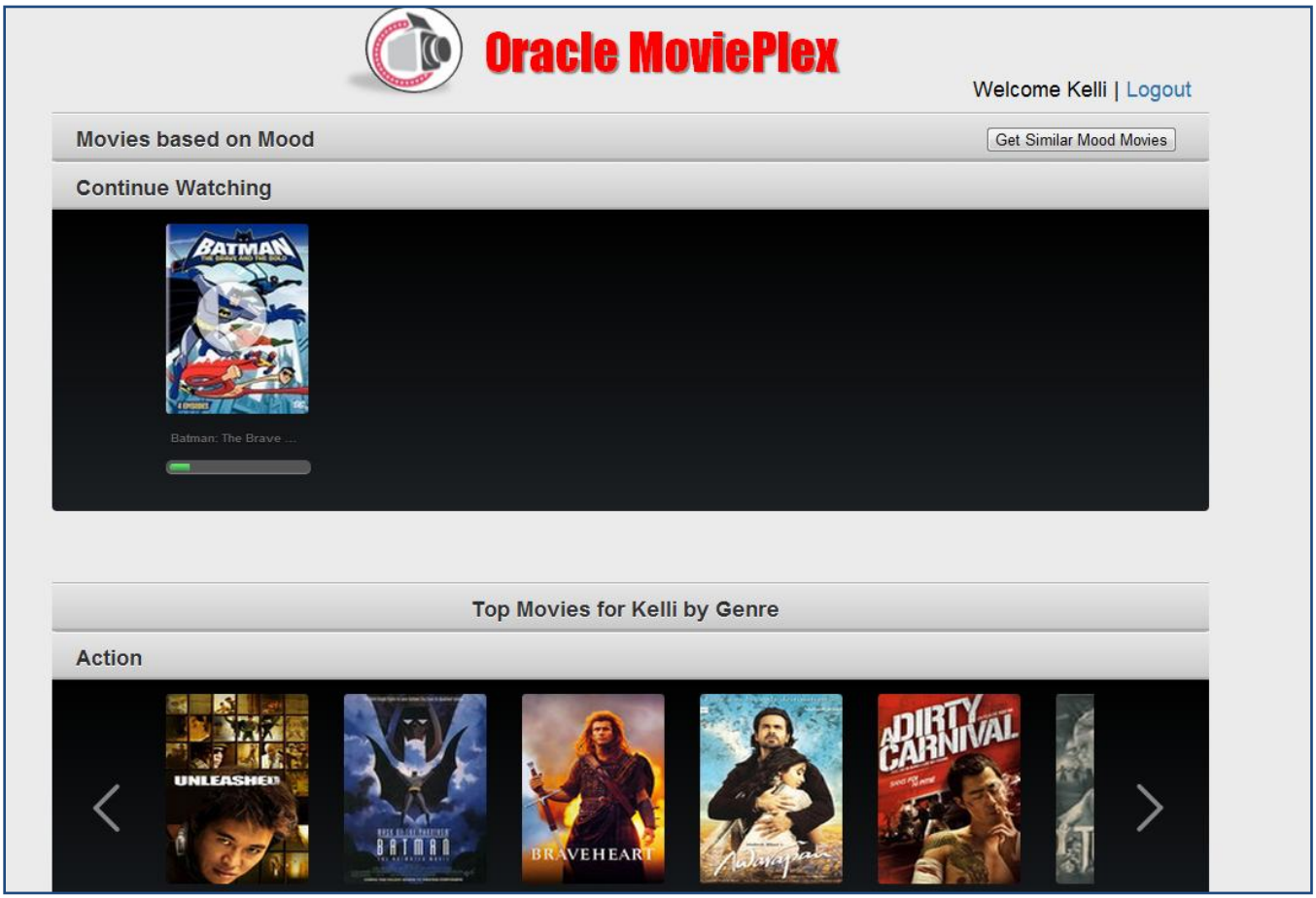


9. Click on the movie that you just selected, which appears in the "Recently Browsed" section.

10. When the description appears, click on the arrow that is visible in the middle of the movie image.

11. Watch the movie for 10 seconds.

12. Click the stop button and close the window. Notice the change in status (green status bar).


Oracle MoviePlex
Welcome Kelli | Logout
Movies based on Mood
Get Similar Mood Movies
Continue Watching
Batman: The Brave ...
Top Movies for Kelli by Genre
Action
UNLEASHED
BATMAN
BRAVEHEART
A DIRTY CARNIVAL

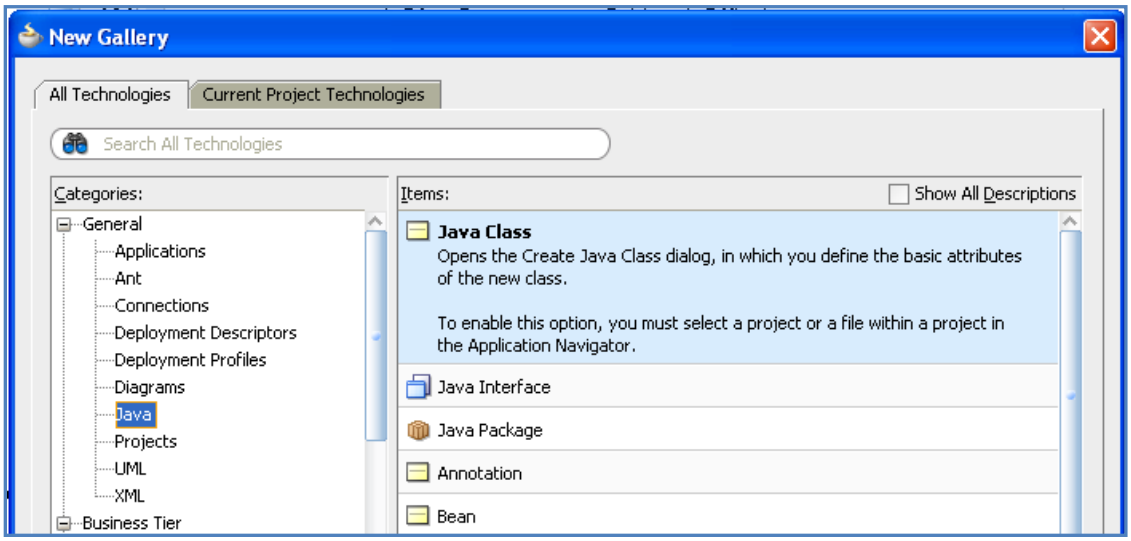
Lab Exercise 4 (Optional) – Query Movie Data

In this exercise, you will query movie data by providing the movieId.

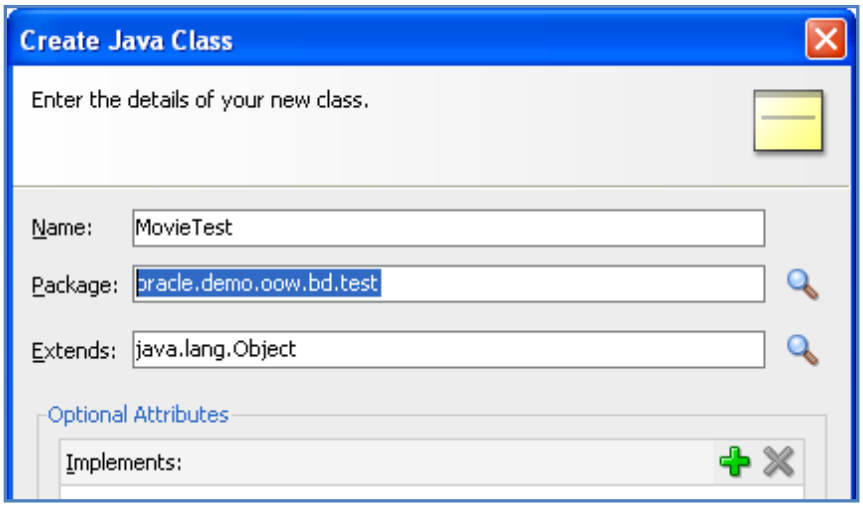
Instructions:

1. Create a new Java class. Select **File > New** from the drop down menu.

2. Select **Java > Java Class** from the dialog box.



3. Type new class-name (**MovieTest**) and the package name (**oracle.demo.oow.bd.test**) and hit **OK** button.

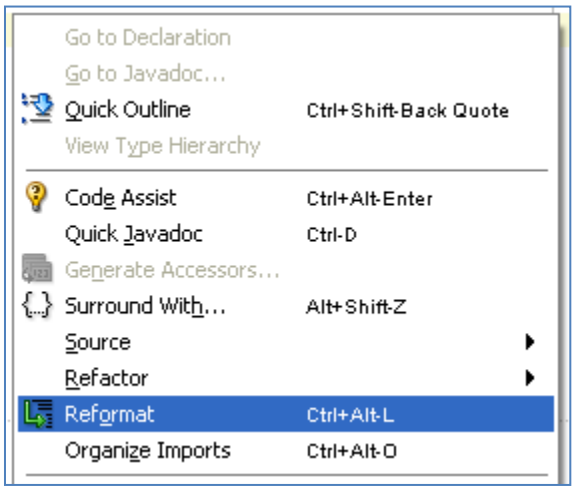


4. Let's create a method *getMovieById(int movieId)* that takes movieId as an input argument, as well as main method that will call *getMovieById* method by passing a movieId. This is how your class should look like at the end.

```
package oracle.demo.oow.bd.test;
import java.util.ArrayList;
import java.util.List;
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv.ValueVersion;

public class MovieTest {
   public MovieTest() {
      super();
   }
   public void getMovieById(int movieId) {
     /** TODO - Code to fetch data from KV-Store will go here **/
   }
   public static void main(String[] args) {
      MovieTest movieTest = new MovieTest();
      movieTest.getMovieById(829);
   }
}
```

Note: When you copy paste the code from document into the JDev, the code might lose the indentation. To indent the code, right-click in JDev main-pain and select *Reformat*



5. Now let's create a connection to a running instance of kvstore, identified by name 'kvstore', and listening at port 5000. Add the code right bellow the /** TODO - Code to fetch data from KV-Store will go here --**/

```
Key key = null;
KVStore kvStore = null;


try {
  if (movieId > -1) {
    /**
    * Create a handle to the kvStore, so we can run get/put operations
    */
    kvStore =
        KVStoreFactory.getStore(new KVStoreConfig("kvstore",
"localhost:5000"));


    /** TODO – Add code here to create a KEY **/


  }
} catch (Exception e) {
  System.out.println("ERROR: Please make sure Oracle NoSQL Database is
running");
}
```

6. Next, create a Key by setting a prefix 'MV' and movieId as the major components to the Key. Add the code right bellow the /** TODO – Add code here to create a KEY --**/

```
/**
* Create a key to fetch movie JSON from the kv-store.
* In our case we saved movie information using this key structure:
* /MV/movieId/ .
*/
List<String> majorComponent = new ArrayList<String>();
majorComponent.add("MV");
majorComponent.add(Integer.toString(movieId));
key = Key.createKey(majorComponent);
/** TODO – Add code to execute get operation **/
```

7. Once we defined the key, next step is to run the get operation by passing the key. Add this code right bellow /** TODO – Add code to execute get operation **/

```
/**
* Now run a get operation, which will return you ValueVersion
* object.
*/
ValueVersion valueVersion = kvStore.get(key);
/** TODO – Add code to display the Value (i.e. movie-information) **/
```

8. Let's now display the Value, which should be the JSON representation of the movie details. Paste this code right bellow /** TODO – Add code to display the Value .. **/

```
/**
* If ValueVersion is not null, get the Value part and convert
* the Byte[] into String object.
*/
if (valueVersion != null) {
   Value value = valueVersion.getValue();
   String movieJsonTxt = new String(value.getValue());
   System.out.println(movieJsonTxt);
} //if (valueVersion != null)
```

9. Now run your program from the JDev by right clicking in the main pane and selecting the green play icon. What do you see on the stdout?

{"id":829,"original_title":"Chinatown","release_date":"1974","overview":"JJ 'Jake' Gittes is a private detective who seems to specialize in matrimonial cases. He is hired by Evelyn Mulwray when she suspects her husband Hollis, builder of the city's water supply system, of having an affair. Gittes does what he does best and photographs him with a young girl but in the ensuing scandal, it seems he was hired by an impersonator and not the real Mrs. Mulwray. When Mr. Mulwray is found dead, Jake is plunged into a complex web of deceit involving murder, incest and municipal corruption all related to the city's water supply.","vote_count":110050,"popularity":8.4,"poster_path":"/6ybT8RbSbd4AltIDABuv39dgqMU.jpg","runtime":0,"genres":[{"id":"8","name":"Crime"},{"id":"3","name":"Drama"},{"id":"20","name":"Mystery"},{"id":"9","name":"Thriller"}]}