

# Testcase “User Administration” Documentation

## Creating the Database

Before starting any execution, we need to set up our database, for that we need to download and install PostgreSQL 11.x from its official website: <https://www.postgresql.org/download/>

We go through the installation accepting all the default steps. Once installed, we run the psql command line console and connect with the localhost server and the default superuser.

First, we are going to create a new database called bank\_user\_administration and connect to it:

```
CREATE DATABASE bank_user_administration;  
\c bank_user_administration
```

We are going to create the users that are going to be the administrators of the new database, to manage the permissions of those users, we will create a role and assign the users to that role, so we can just set the permissions to it and they will be valid for all the users.

```
create role bankadmin1 login password 'admin';  
create role bankadmin2 login password 'admin';  
create role bankadmin3 login password 'admin';  
create role bank_administrator;  
grant bank_administrator to bankadmin1;  
grant bank_administrator to bankadmin2;  
grant bank_administrator to bankadmin3;
```

Now we are going to revoke all the permissions for the public access (any new user) to the database and grant only to the bank\_administrator group, so we can assure that nobody else can access the database:

```
revoke all on database bank_user_administration from public;  
grant connect on database bank_user_administration to  
bank_administrator;
```

We need to create two tables for our test case, one for the users of the bank, and another for the accounts. Those tables are going to be related with OneToMany relationship. It means that one user can have multiple accounts, but each account can only belong to one user. To create them we type the next code:

```
create table public.bankuser(  
user_id serial,  
first_name varchar(255) not null,  
last_name varchar(255) not null,  
owner varchar(255) not null default session_user,  
primary key (user_id)  
);
```

```

create table public.account(
account_id serial,
iban varchar(22) not null unique,
owner varchar(255) not null default session_user,
primary key (account_id),
fk_user_id int not null,
foreign key (fk_user_id) references public.bankuser(user_id)
);

```

We have just created the basic fields so we can maintain the simplicity. We have added id fields to each table, to act as primary keys and type “serial”, so they are going to be auto incremental. We have also created a default field “owner” in each table, filled by default with the user who create the entry, so we can later configure the permissions for each table.

Now we set the permissions over the table for the admin user group, and grant usage over the automatically created sequences (necessary to auto increment the ids). We are only going to assign insert and update permission to the editable fields, the rest of the fields doesn’t need those permissions because there are automatic fields.

```

grant select, insert(user_id, first_name, last_name), update(first_name,
last_name), delete on public.bankuser to bank_administrator;

```

```

grant usage on sequence bankuser_user_id_seq to bank_administrator;

```

```

grant select, insert(account_id, iban, fk_user_id), update(iban), delete
on public.account to bank_administrator;

```

```

grant usage on sequence account_account_id_seq to bank_administrator;

```

Finally, we are going to use PostgreSQL functionality to create specific rules to restrict the update and delete permissions over the tables, to the administrator who created each entry. For that we are going to use policies:

```

create policy bankuser_select on public.bankuser
for select to bank_administrator
using (true);

```

```

create policy bankuser_insert on public.bankuser
for insert to bank_administrator
with check (owner = session_user);

```

```

create policy bankuser_update on public.bankuser
for update to bank_administrator
using (owner = session_user);

```

```

create policy bankuser_delete on public.bankuser
for delete to bank_administrator
using (owner = session_user);

```

```

create policy account_select on public.account
for select to bank_administrator
using (true);

```

```
create policy accountuser_insert on public.account
for insert to bank_administrator
with check (owner = session_user);
```

```
create policy account_update on public.account
for update to bank_administrator
using (owner = session_user);
```

```
create policy account_delete on public.account
for delete to bank_administrator
using (owner = session_user);
```

Once the policies are enable, we need to activate them. For that we enable the row level security for the tables affected:

```
alter table public.bankuser enable row level security;
alter table public.account enable row level security;
```

We are now ready to test our application.

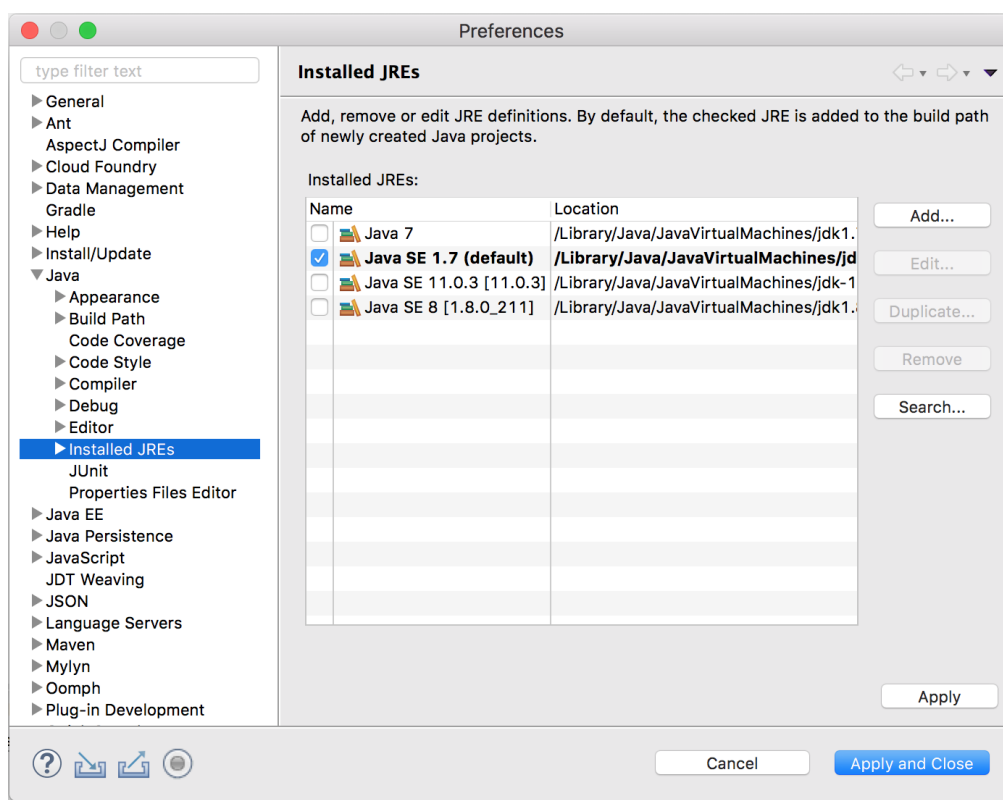
## Preparing the execution environment

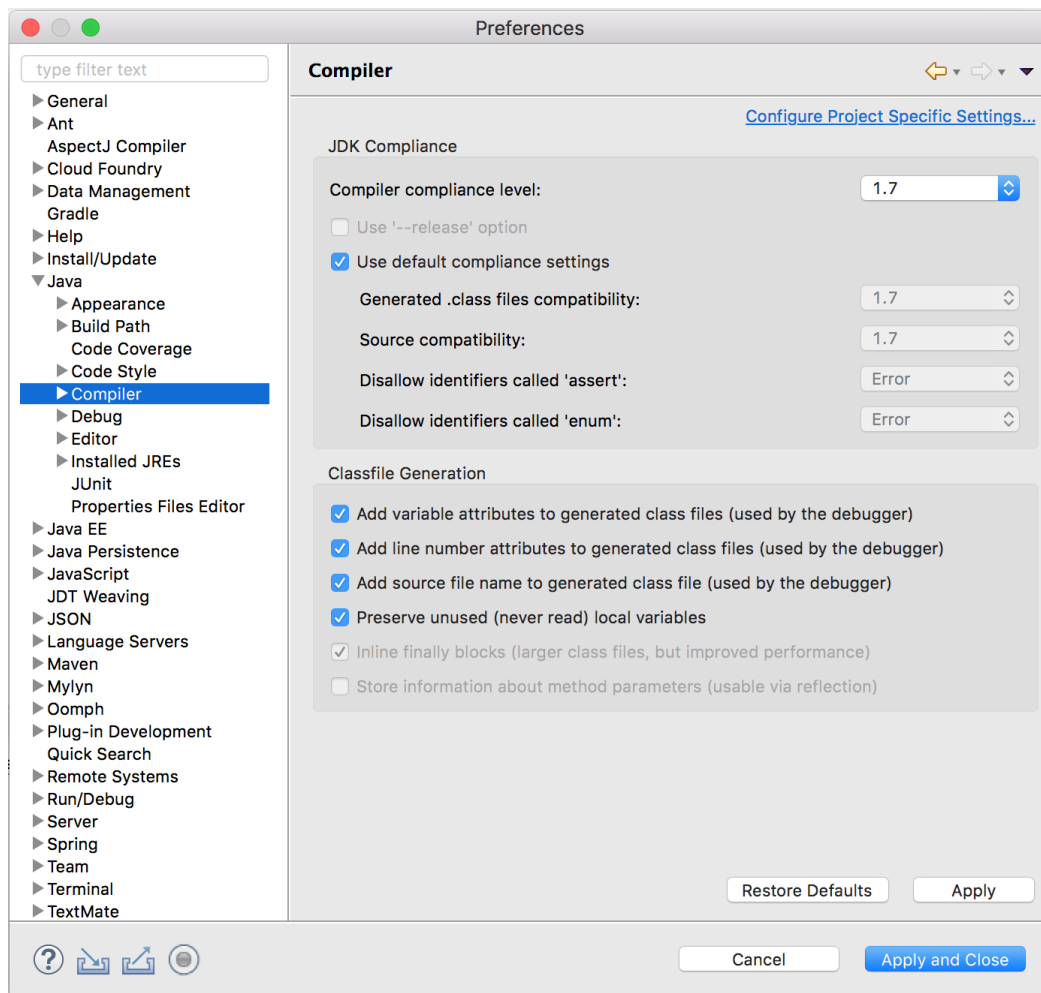
As execution and testing environment we are going to use Eclipse. To install this software we need to have a Java JDK installed in our system

(<https://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html>). Then we can install Eclipse from his official page (<https://www.eclipse.org/downloads/>).

Once downloaded and installed we are going to configure it to run with the Java JDK 7.

For that, we are going to enter into the preferences (Window/Preferences on Windows. Eclipse menu/Preferences on Mac OSX) Java/Installed JREs section and add our downloaded JDK as default. Then into Java/Compiler, and set it to level 1.7.





Now we are going to clone the git repository into our system.

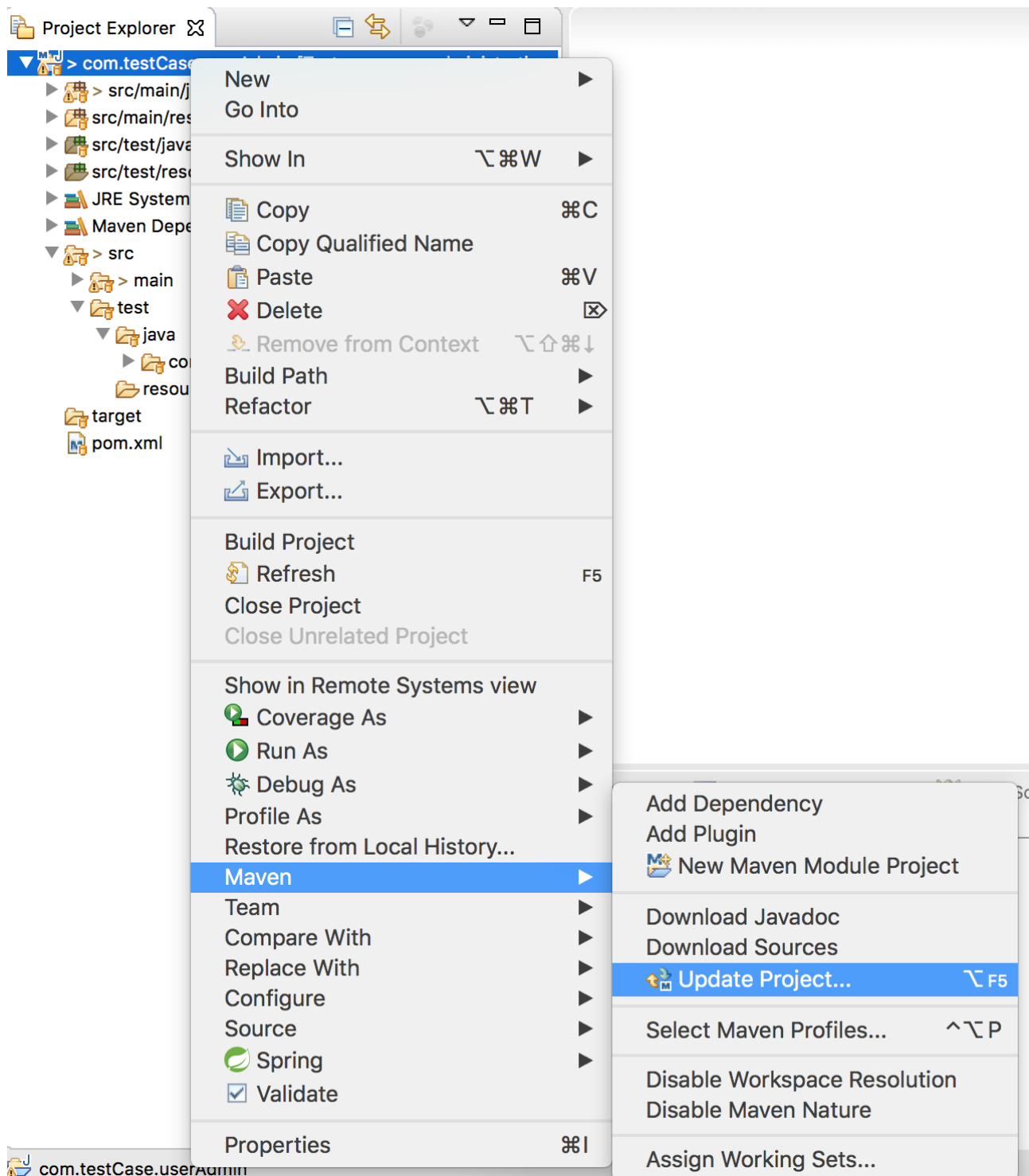
This is the repository for the project: <https://github.com/Aslat/Testcase-user-administration>

To clone it we need to install Git in our system, access through command line to the folder where we want to clone the project, and type:

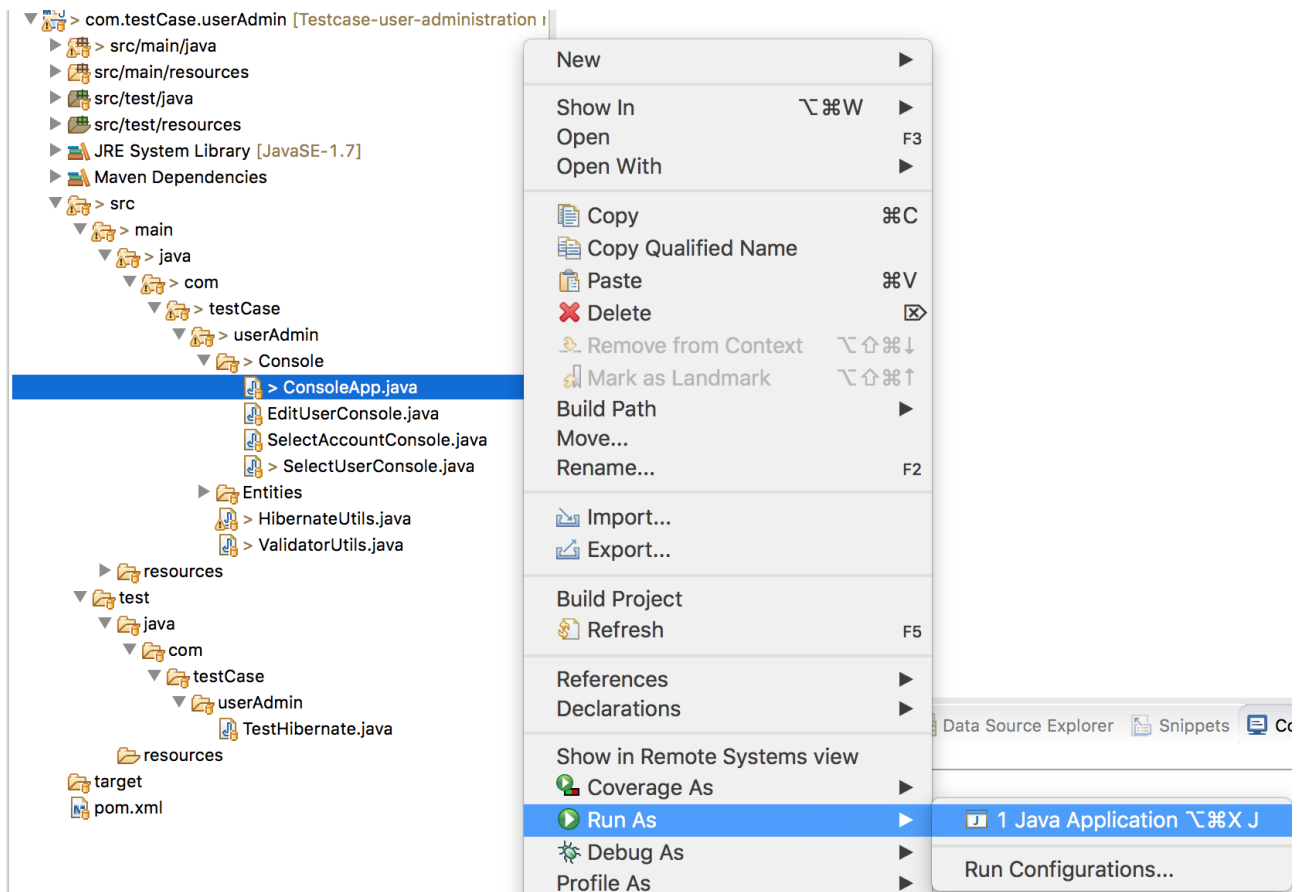
git clone <https://github.com/Aslat/Testcase-user-administration.git>

Now we have to add the project into Eclipse: For that we click File/Import/Existing Maven Projects and select the path where we have cloned the project.

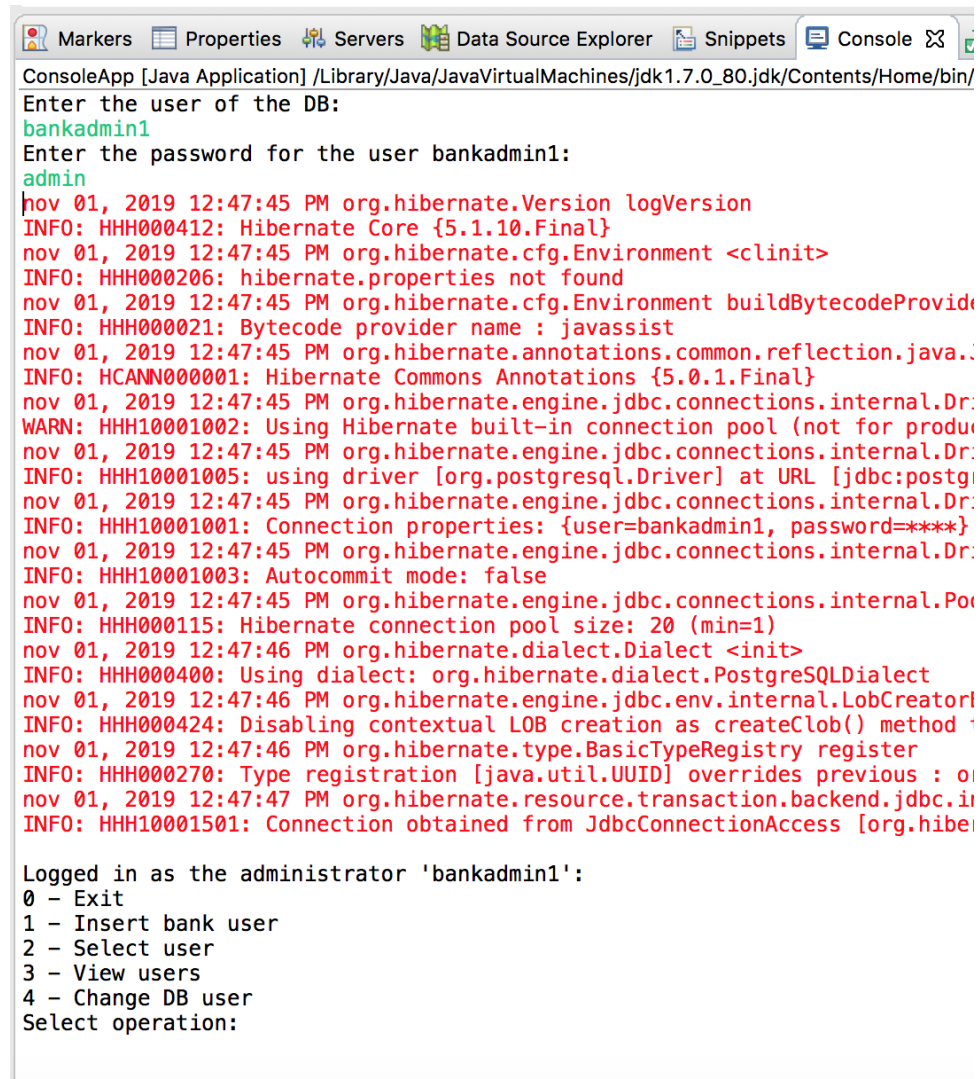
Once imported, as we are using maven to manage all the project dependencies, we are going to update the project so maven can download all the dependencies. Right click on the project, Maven/Update Project



Finally, to run the console Application we have to open the folder structure of the project to src/main/java/com/testCase/userAdmin/Console/ConsoleApp.java, right click on this file and select Run As/Java Application.



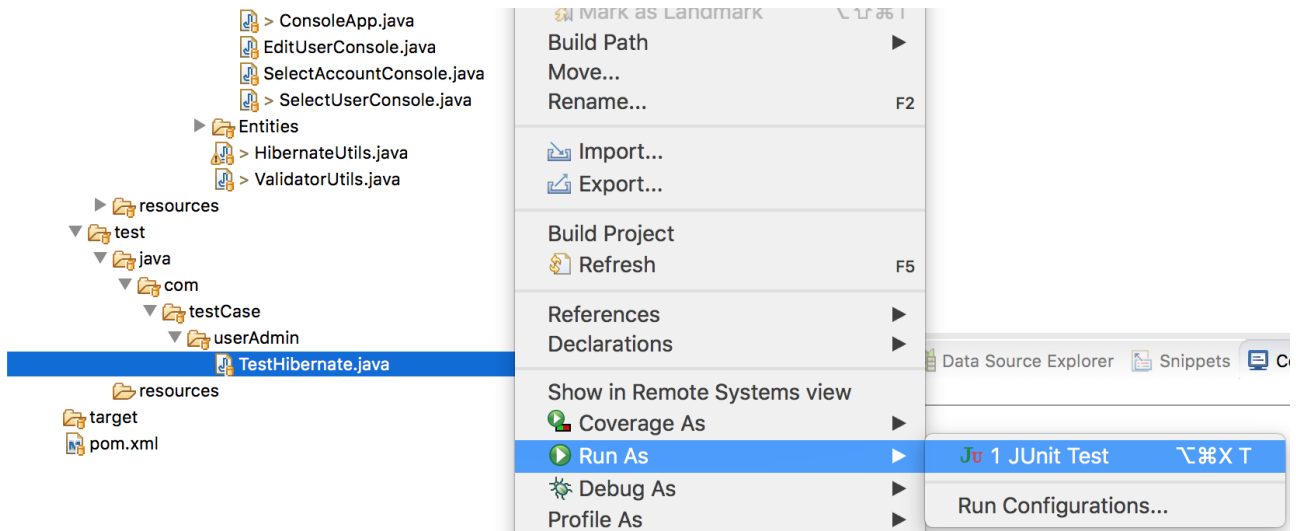
The options available to connect to the DB are the superuser used to create the DB at the beginning of the documentation (in my case, user: postgres, password:admin) or use the administrator users created to have the specific access and permissions over the DB: bankadmin1, bankadmin2 and bankadmin3. All of them have the same password: admin.



```
ConsoleApp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_80.jdk/Contents/Home/bin/
Enter the user of the DB:
bankadmin1
Enter the password for the user bankadmin1:
admin
nov 01, 2019 12:47:45 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {5.1.10.Final}
nov 01, 2019 12:47:45 PM org.hibernate.cfg.Environment <clinit>
INFO: HHH000206: hibernate.properties not found
nov 01, 2019 12:47:45 PM org.hibernate.cfg.Environment buildBytecodeProvider
INFO: HHH00021: Bytecode provider name : javassist
nov 01, 2019 12:47:45 PM org.hibernate.annotations.common.reflection.java.CommonReflectionCache <init>
INFO: HCAN000001: Hibernate Commons Annotations {5.0.1.Final}
nov 01, 2019 12:47:45 PM org.hibernate.engine.jdbc.connections.internal.Drivers <init>
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production)
nov 01, 2019 12:47:45 PM org.hibernate.engine.jdbc.connections.internal.Drivers <init>
INFO: HHH10001005: using driver [org.postgresql.Driver] at URL [jdbc:postgresql://localhost:5432/bankdb]
nov 01, 2019 12:47:45 PM org.hibernate.engine.jdbc.connections.internal.Drivers <init>
INFO: HHH10001001: Connection properties: {user=bankadmin1, password=admin}
nov 01, 2019 12:47:45 PM org.hibernate.engine.jdbc.connections.internal.Drivers <init>
INFO: HHH10001003: Autocommit mode: false
nov 01, 2019 12:47:45 PM org.hibernate.engine.jdbc.connections.internal.PoolingDelegate <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
nov 01, 2019 12:47:46 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
nov 01, 2019 12:47:46 PM org.hibernate.engine.jdbc.env.internal.LobCreatorImpl <init>
INFO: HHH000424: Disabling contextual LOB creation as createClob() method is deprecated
nov 01, 2019 12:47:46 PM org.hibernate.type.BasicTypeRegistry register
INFO: HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDType
nov 01, 2019 12:47:47 PM org.hibernate.resource.transaction.backend.jdbc.internal.JdbcTransactionImpl <init>
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.connections.internal.PoolingDelegate]

Logged in as the administrator 'bankadmin1':
0 - Exit
1 - Insert bank user
2 - Select user
3 - View users
4 - Change DB user
Select operation:
```

To run the JUnit test, we have to open the folder structure of the project to `src/test/java/com/testCase/userAdmin/TestHibernate.java`, right click on this file and select Run As/JUnit Test.



To see the result of the tests, we have to open the tab JUnit:

