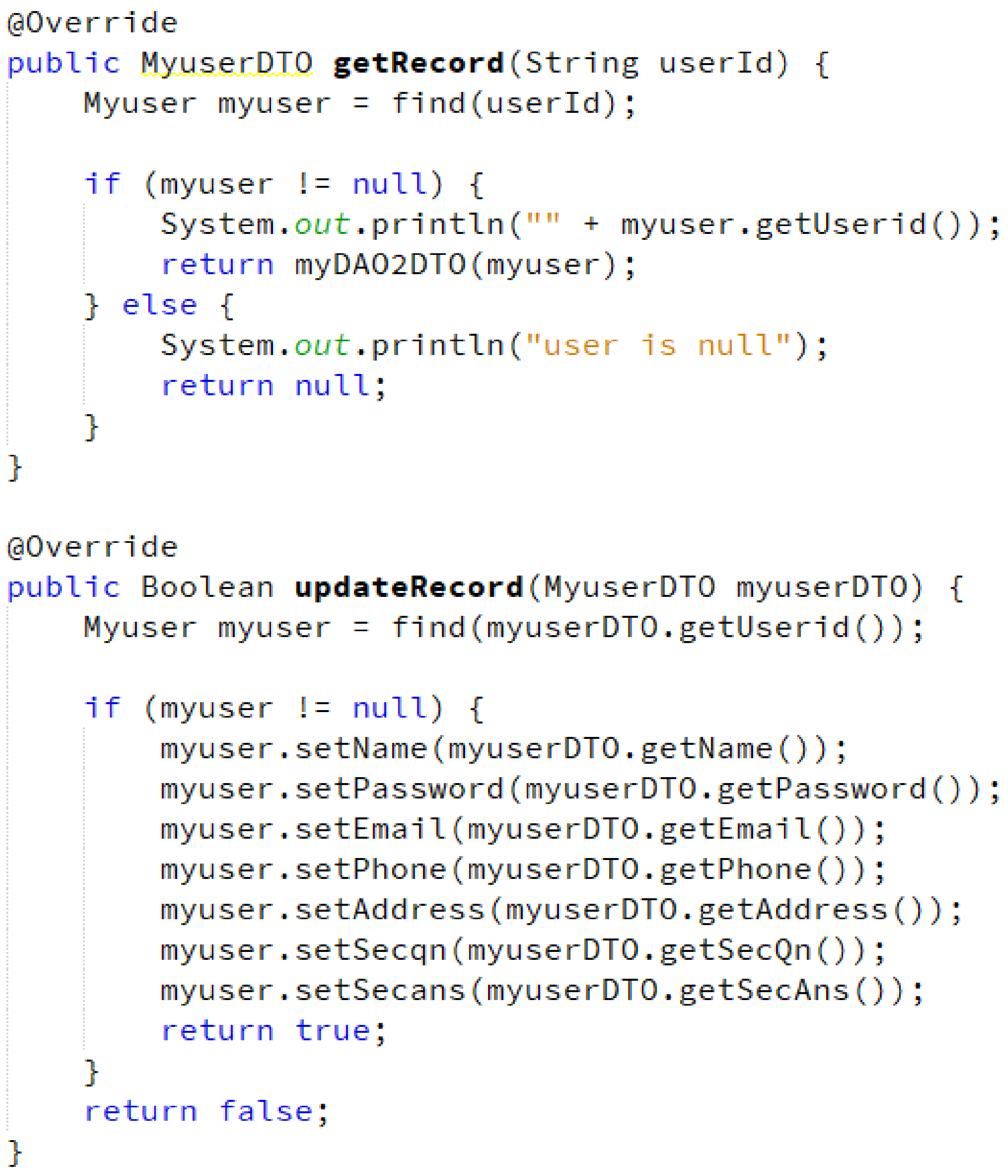
4.1P

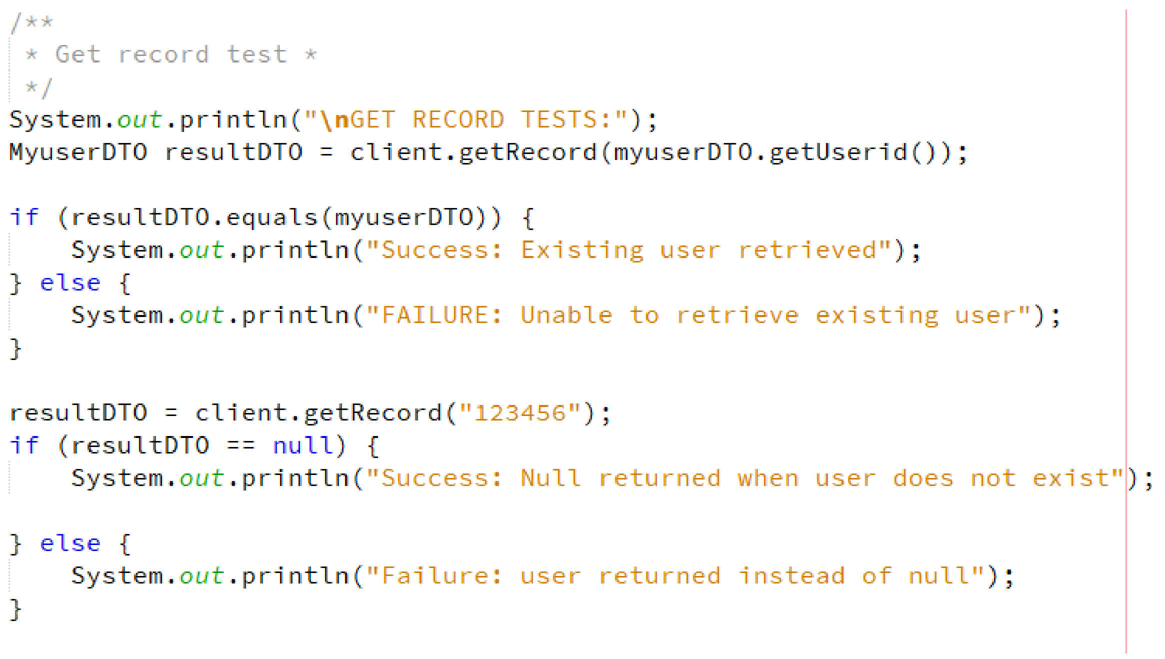
1. Crud operations in MyuserFacade



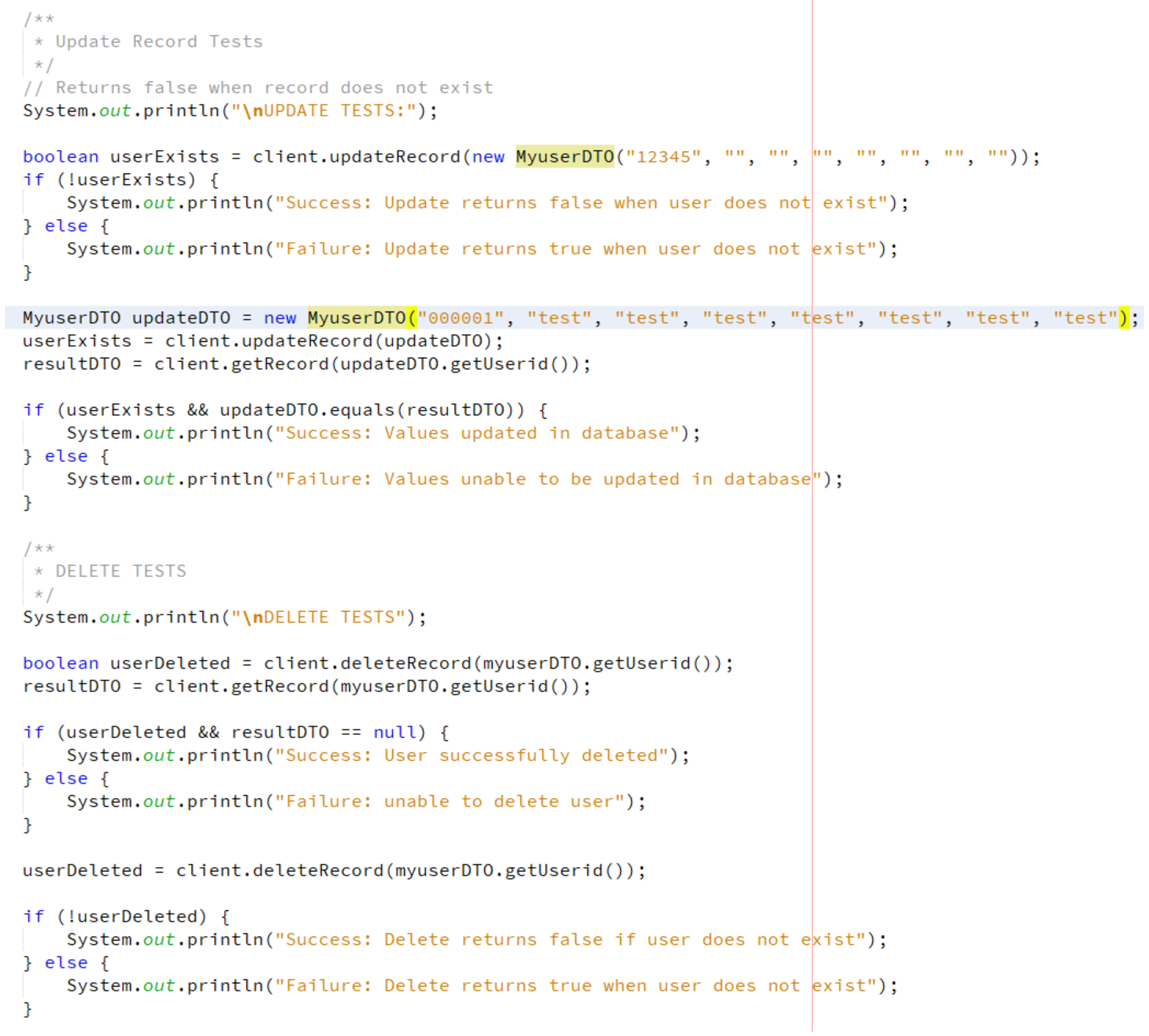
```
@Override
public MyuserDTO getRecord(String userId) {
    Myuser myuser = find(userId);

    if (myuser != null) {
        System.out.println("" + myuser.getUserid());
        return myDAO2DTO(myuser);
    } else {
        System.out.println("user is null");
        return null;
    }
}

@Override
public Boolean updateRecord(MyuserDTO myuserDTO) {
    Myuser myuser = find(myuserDTO.getUserid());

    if (myuser != null) {
        myuser.setName(myuserDTO.getName());
        myuser.setPassword(myuserDTO.getPassword());
        myuser.setEmail(myuserDTO.getEmail());
        myuser.setPhone(myuserDTO.getPhone());
        myuser.setAddress(myuserDTO.getAddress());
        myuser.setSecqn(myuserDTO.getSecQn());
        myuser.setSecans(myuserDTO.getSecAns());
        return true;
    }
    return false;
}
```

```java
@Override
public Boolean deleteRecord(String userId) {
    Myuser myuser = find(userId);

    if (myuser != null) {
        remove(myuser);
        return true;
    }

    return false;
}

@Override
public ArrayList<MyuserDTO> getRecordsByAddress(String address) {

    List<Myuser> queryResult = em.createQuery(
            "Select u from Myuser u Where u.address= :uAddress")
            .setParameter("uAddress", address)
            .getResultList();

    ArrayList<MyuserDTO> result = new ArrayList();
    for (int i = 0; i < queryResult.size(); i++)
    {
        result.add(myDAO2DTO(queryResult.get(i)));
    }

    return result;
}
```

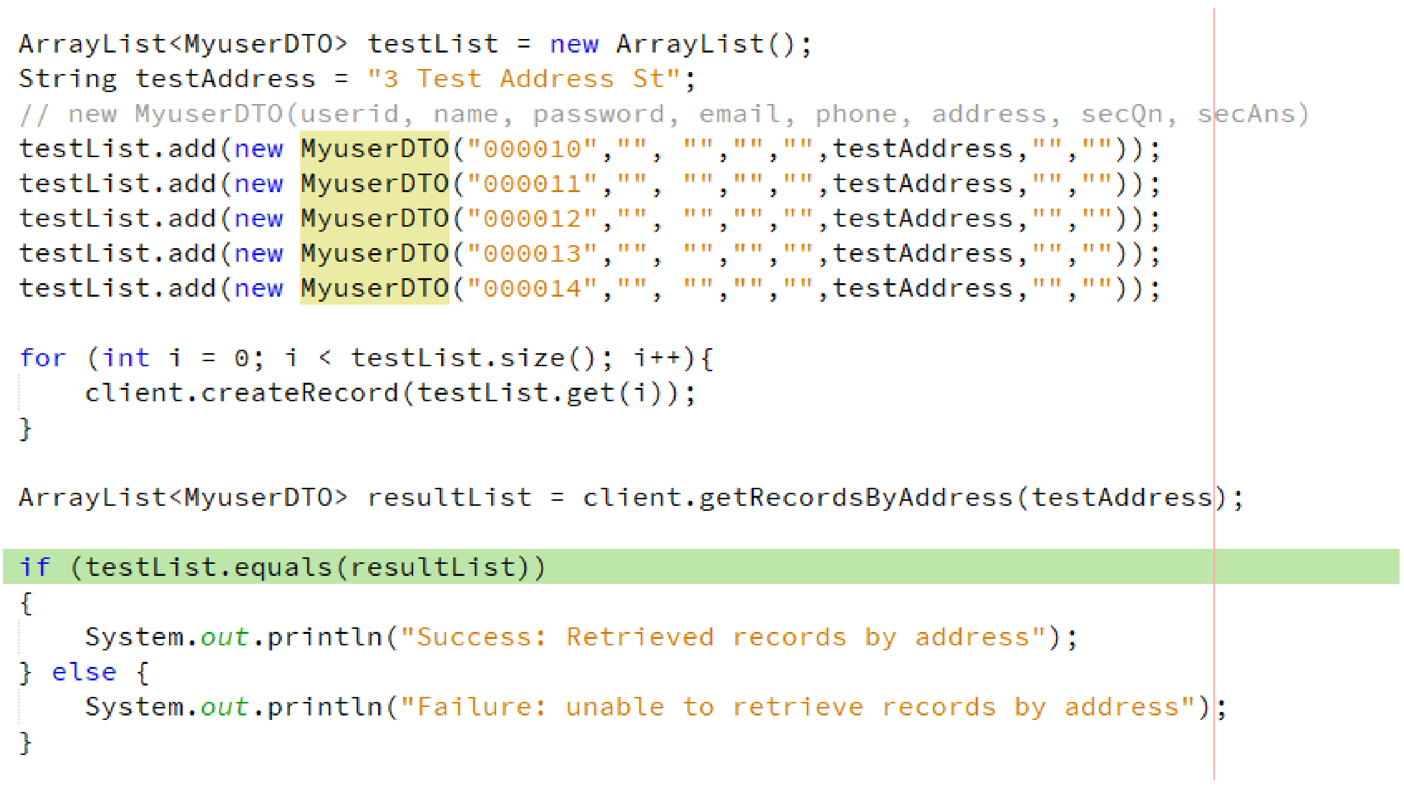
2. Testing



```java
/**
 * Get record test *
 */
System.out.println("\nGET RECORD TESTS:");
MyuserDTO resultDTO = client.getRecord(myuserDTO.getUserid());

if (resultDTO.equals(myuserDTO)) {
    System.out.println("Success: Existing user retrieved");
} else {
    System.out.println("FAILURE: Unable to retrieve existing user");
}

resultDTO = client.getRecord("123456");
if (resultDTO == null) {
    System.out.println("Success: Null returned when user does not exist");

} else {
    System.out.println("Failure: user returned instead of null");
}
```



```java
/**
 * Update Record Tests
 */
// Returns false when record does not exist
System.out.println("\nUPDATE TESTS:");

boolean userExists = client.updateRecord(new MyuserDTO("12345", "", "", "", "", "", "", ""));
if (!userExists) {
    System.out.println("Success: Update returns false when user does not exist");
} else {
    System.out.println("Failure: Update returns true when user does not exist");
}

MyuserDTO updateDTO = new MyuserDTO("000001", "test", "test", "test", "test", "test", "test", "test");
userExists = client.updateRecord(updateDTO);
resultDTO = client.getRecord(updateDTO.getUserid());

if (userExists && updateDTO.equals(resultDTO)) {
    System.out.println("Success: Values updated in database");
} else {
    System.out.println("Failure: Values unable to be updated in database");
}

/**
 * DELETE TESTS
 */
System.out.println("\nDELETE TESTS");

boolean userDeleted = client.deleteRecord(myuserDTO.getUserid());
resultDTO = client.getRecord(myuserDTO.getUserid());

if (userDeleted && resultDTO == null) {
    System.out.println("Success: User successfully deleted");
} else {
    System.out.println("Failure: unable to delete user");
}

userDeleted = client.deleteRecord(myuserDTO.getUserid());

if (!userDeleted) {
    System.out.println("Success: Delete returns false if user does not exist");
} else {
    System.out.println("Failure: Delete returns true when user does not exist");
}
```



```java
ArrayList<MyuserDTO> testList = new ArrayList();
String testAddress = "3 Test Address St";
// new MyuserDTO(userid, name, password, email, phone, address, secQn, secAns)
testList.add(new MyuserDTO("000010","", "","","",testAddress,"",""));
testList.add(new MyuserDTO("000011","", "","","",testAddress,"",""));
testList.add(new MyuserDTO("000012","", "","","",testAddress,"",""));
testList.add(new MyuserDTO("000013","", "","","",testAddress,"",""));
testList.add(new MyuserDTO("000014","", "","","",testAddress,"",""));

for (int i = 0; i < testList.size(); i++){
    client.createRecord(testList.get(i));
}

ArrayList<MyuserDTO> resultList = client.getRecordsByAddress(testAddress);

if (testList.equals(resultList))
{
    System.out.println("Success: Retrieved records by address");
} else {
    System.out.println("Failure: unable to retrieve records by address");
}
```

3. Questions

1. Who is doing the ORM work in this project?

The ORM work will always be performed by the entity classes that map their properties onto the database. As a result the Myuser class is performing the ORM work here alongside the java persistence library.

Intuitively you may say that MyuserFacade is performing the work as it defines the methods for interacting with the database, however these are actually just business methods, as the interaction with the database is done indirectly via the Entity manager class provided by the java persistence library.

2. Explain (in your own words) the concept of bean instance pooling in the context of stateless session bean using “MyuserFacade” as an example. Also explain how it can achieve scalability.

Bean instance pooling is when the server creates a number of bean instances and stores them in a cache. When a client makes a call to the remote interface the server binds a bean to the clients request and uses that object to perform the function. So in 4.1 the MyuserAppClient makes a call to the remote interface provided by ED-JEE-DTO, in this case being MyuserFacade. The existing session bean is then bound to the client and performs the function requested and is then freed up for the next client request.

Bean pooling improves scalability as a small number of beans can be used to serve many clients, whereas creating a bean for each client would consume much more cpu execution time and memory.