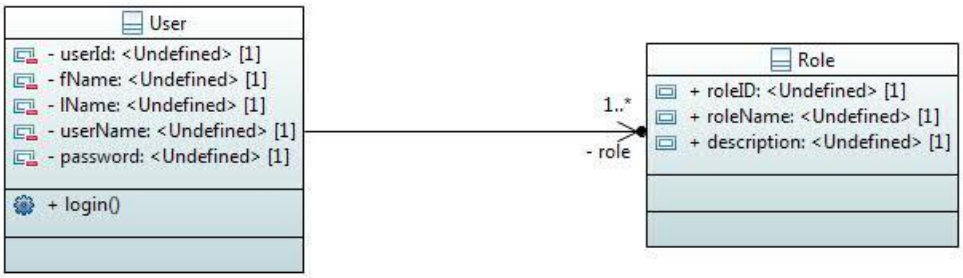
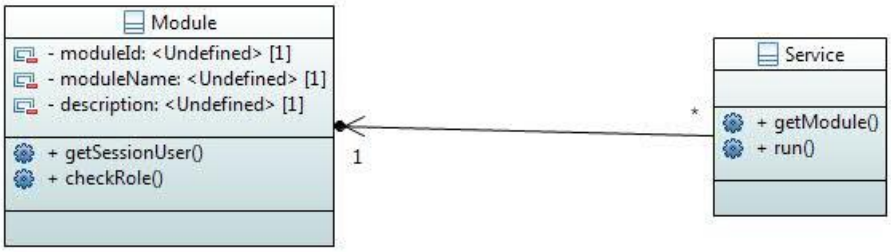
Answer 1:

```
private Optional<Integer> getId(int type, String appId, String openId) {

    Integer id = null;
    switch (type) {

        case 1:
            Type1User type1User = userRepository.getType1User(openId);
            validateUserNotNull(type1User);
            id = type1User.getId();
            break;
        case 2:
            Type2User type2User = userRepository.getType2User(openId);
            validateUserNotNull(type2User);
            id = type2User.getId();
            break;
        case 4:
            Type4User type4User =
userRepository.getType4UserByOpenId(openId);
            validateUserNotNull(type4User);
            id = type4User.getId();
            break;
        case 5:
            Type5User type5User =
userRepository.getType5UserByOpenId(openId);
            validateUserNotNull(type5User);
            id = type5User.getId();
            break;
        case 6:
            Type6User type6User =
userRepository.getType6UserByOpenId(openId);
            validateUserNotNull(type6User);
            id = type6User.getId();
            break;
        case 8:
            Type8User type8User =
userRepository.getType8User(Long.valueOf(openId));
            validateUserNotNull(type8User);
            id = type8User.getId();
            break;
        case 11:
            Type11User type11User =
userRepository.getType11UserByTokenAndUID(appId, openId);
            validateUserNotNull(type11User);
            id = type11User.getId();
            break;
        case 12:
            Type12User type12User =
userRepository.getType12UserByAppIdAndOpenId(appId, openId);
            validateUserNotNull(type12User);
            id = type12User.getId();
            break;
        case 13:
            Type13User type13User =
userRepository.getType13UserByAppIdAndOpenId(appId, openId);
            validateUserNotNull(type13User);
            id = type13User.getId();
```

```
        default:
            break;
    }
    return Optional.ofNullable(id);
}

private void validateUserNotNull(Object user) {

    if (user == null) {
        throw new UserNotFoundException();

   }
}
```

Ans : 2 Object Oriented diagram




```
Ans 3: public Customer findCustomerByName(String customerName) {
  try {
    Customer c = customerService.findByName(customerName);
    return c;
  } catch (Exception ex) {
    LOG.error("Exception looking up customer by name: " + ex.getMessage(),
ex);
  }
  return null;
}
```

```
User findByUsername(String userName) throws UserNameNotFoundException {
  EntityManager em = entityManagerFactory.createEntityManager();
  return em.createQuery("from User where userName = :userName", User.class)
  .setParameter("userName", userName)
  .getSingleResult();
}

public String login(Model model, String username, String password) {
  try {
    // attempt to login user
    userService.login(username, password);
  } catch (Exception ex) {
    model.addAttribute("error", ex.getMessage());
  }
  return "login”;
}
```

Ans:4

```
User findByUsername(String userName) throws UserNameNotFoundException {
  EntityManager em = entityManagerFactory.createEntityManager();
  return em.createQuery("from User where userName = :userName", User.class)
  .setParameter("userName", userName)
  .getSingleResult();
}
```

Ans:5 @Path(“/”)

```
@PermitAll

public class userExample {

 @DELETE

 @Path(“users”)

 @Produces(“text/plain”)

 @RolesAllowed({“Admin”, “Manager”})

 public String deleteAllUsers() {

  return userService.deleteAllUsers();

 }



 @GET

 @Path(“users”)
```

```
 @Produces(“text/plain”)
 public String getAllUsers() {
  return userService.getAllUsers();
 }
}
```

Ans 6:Unit Test

```
public class ShowQuestionsActivityTest extends
        ActivityInstrumentationTestCase2<ShowQuestionsActivity> {

    public ShowQuestionsActivityTest() {
        super(ShowQuestionsActivity.class);
    }

    public void testHello() {
        fail("Not yet implemented");
    }
}

Open the coverage report (use open, xdg-open,
public class ShowQuestionsActivityTest extends
ActivityInstrumentationTestCase2<ShowQuestionsActivity> {
    private Solo solo;

    public ShowQuestionsActivityTest() {
        super(ShowQuestionsActivity.class);
    }

    @Override
    protected void setUp() {
        solo = new Solo(getInstrumentation(), getActivity());
    }

    public void testShowQuestion() {
        assertTrue("Question is displayed", solo.searchText("What is the
answer to Life, the universe and everything?"));
        assertTrue("Correct answer is displayed", solo.searchText("Forty-
two"));
        assertTrue("Incorrect answer is displayed", solo.searchText("Twenty-
seven"));

        Button nextQuestionButton = solo.getButton("Next question");
        assertFalse("Next question button is disabled",
nextQuestionButton.isEnabled());
   }
}
Ans : 7 $ cd ${REPO}
$ ${ANDROID_SDK}/tools/android update project --path SwEng2013QuizApp
$ ${ANDROID_SDK}/tools/android update project --path SwEng2013QuizAppTest
```

```
$ ${ANDROID_SDK}/tools/android update test-project --path
SwEng2013QuizAppTest --main ../SwEng2013QuizApp
# start an emulator from the terminal (or from Eclipse, if you prefer)
# SwEngAndroidDevice is the name of your Android Virtual Device, as per the
Android tutorial
$ ${ANDROID_SDK}/tools/emulator -avd SwEngAndroidDevice
$ cd ${REPO}/SwEng2013QuizAppTest
$ ant clean emma debug install test
```