# Veri Tabanı Yönetimi Lab#9

->Öncelikle client ve server arasında bağlantı kurulur. Bağlantı kurulduktan sonra client, server'a bilgi isteği gönderir. Server veritabanına erişip bilgiyi çeker; client'a cevap olarak mesaj gönderir. Bağlantı kopana dek client, server'dan çeşitli isteklerde bulunabilir. Server, aynı anda birden çok client'la bağlantı kurabilir.

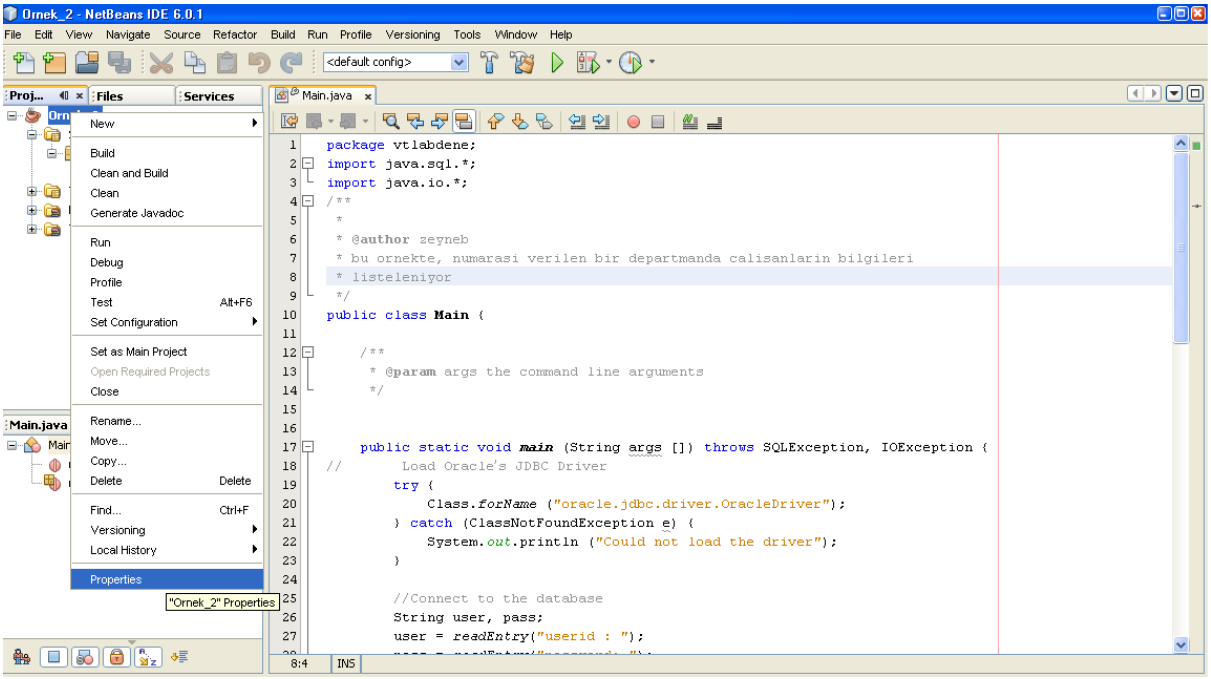
->Bir java client'ı, bir veritabanı server ile JDBC library'si kullanarak iletişim kurar.
JDBC class'ları bir java client'ı ile server arasındaki veri transferini yönetir.

Bu dökumanda, Netbeans'ten PostgreSQL sunucusuna bağlanmak için gereken işlem adımlarının incelenmesi ve basit örneklerin çözümü bulunmaktadır. Öncelikle uygun bir JDBC sürücüsü indirilmelidir. Aşağıdaki adresten bu sürücüyü bulup indirebilirsiniz:
http://jdbc.postgresql.org/download.html

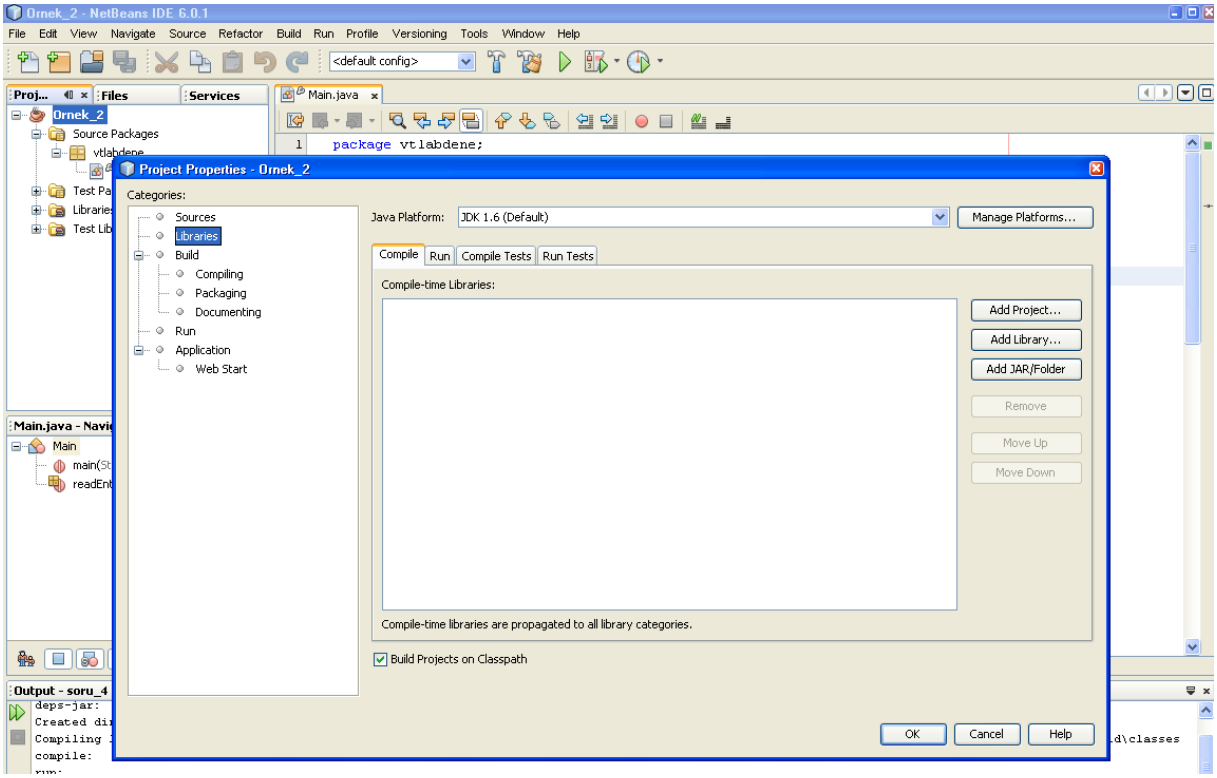
Bu jar dosyası indirildikten sonra java projenizde bu jar'ın path'ini göstermelisiniz. Örneğin Indirilen jar'ın ismi "jdbc" olsun. Bu jar'ı projenize eklemek için gereken işlem adımları:

İlgili işlem adımları:

1. projeyi sağ tıklayıp properties'i seçin:

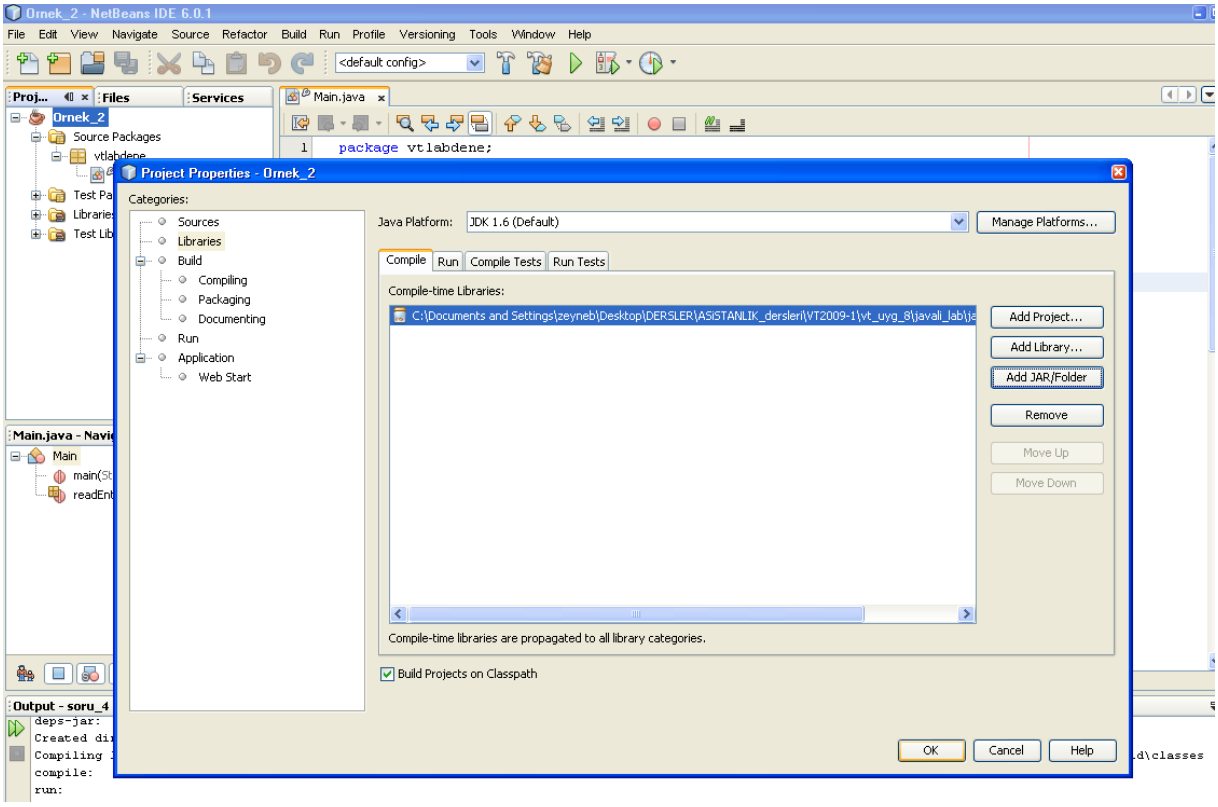



2. Gelen ekranda libraries'i tıklayın




3. Add JAR/Folder'ı seçin:

jdbc.jar'ı ekleyin ve artık "import java.sql.*;" diyerek, veritabanınıza bağlanıp, sql sorgularınızı java'da yazabilirsiniz.




**Örnekler:**

**ÖRNEK - 1:** SSN'i verilen çalışanın soyad ve maaş bilgisini görelim:

**Örnek1'in 20-21 arası satırlarında:**

```
Connection conn =
        DriverManager.getConnection("jdbc:postgresql://localhost:5432/postgres", user, pass);
```

| | |
|---|---|
| “jdbc:postgresql”: | Driver’ın kullanacağı protokolü tanımlar; |
| localhost:5432: | Server olan makinenin adresi, varsa port numarası; |
| vtlab: | Bağlanılan veritabanının ismini. |
| user: | Veritabanındaki kullanıcı ismi |
| pass: | Kullanıcı şifresi |

**Örnek1’in 23-24 arası satırlarında:**

```
String query =
                "select LNAME,SALARY from EMPLOYEE where SSN = ?";
PreparedStatement p = conn.prepareStatement (query);
```

**Sorgu bir statement’a hazırlanır. “?” olan yerde sorguya dışardan bir parametre eklenir. Bu parametreyi kullanıcı dışardan girer:**

```
String ssn = readEntry("Enter a Social Security Number: ");
p.clearParameters();
p.setString(1,ssn);
```

**Statement’taki sorgu çalıştırılır. Çalıştıktan sonra sonucu bir resultset’e atanır:**

```
ResultSet r = p.executeQuery();
```

**Result set’te veri olduğu sürece “r.next()” true döner; veri yoksa false döner:**

```
if (r.next ())
```

**Almak istediğimiz bilgileri resultset’ten istenen formatta çekeriz. (Ister string, ister double):**

```
String lname = r.getString(1);
double salary = r.getDouble(2);
```

**Sorgunun hazırlandığı statement’ı kapatırız (p):**

```
p.close();
```

**Bağlantıyı da kapatırız:**

```
conn.close();
```

**ÖRNEK - 2:** Bir departman no'su verildiğinde, bu departmanda çalışanların soyad ve maaş bilgilerini görelim:

**ÖRNEK2’de ise bir “statement” nesnesi create ederiz: (37.satır)**

```
Statement s = conn.createStatement();
```

Sorguyu bu statement üzerinden çalıştırırız. Yine resultset dolu olduğu sürece ekrana sonuçları yazdırırız. (**satır 40-44 arası**)

Işimiz bitince statement’ı ve bağlantıyı kapatırız.

**Statement ile preparedStatement arasındaki temel fark:**

PreparedStatement’ta query string’i veritabanı sunucusuna gönderilir; sırayla once syntax kontrolu sonra sorgu execute’u yapılır.

Statement’ta ise sorgunun syntax kontrolü ve çalıştırılması aynı anda yapılır.

Ayrıca örnek2’deki gibi, sorgu "Statement" üzerinden yazıldığında once dept no kullanıcıdan alınır; sonra sorgu string’ine bu değer eklenir. Ancak PreparedStatement’ta ise kullanıcıdan

SSN'i alınmadan once zaten sorgu hazırdır. SSN no'su sorguya parameter olarak sonradan eklenir.

PreparedStatement class'ı bir programda, aynı sorgu stringi (sadece bir parameter değişikliği ile) birkaç kez tekrar edecekse kullanılmalıdır.

**Örneklerde incelenen kodların özeti:**

* Öncelikle client ile server arasında bir connection kurmalıyız. (Connection nesnesi ile)
* Connection nesnesinin createStatement metodu, statement tipinde bir nesne döner.
* Statement nesnesinin 2 metodu sqlStatement'ı çalıştırır: executeQuery ve executeUpdate
* executeQuery: arguman olarak sql query'yi alır ve çalıştırır.
* executeUpdate: create table, insert, update, delete gibi komutları çalıştırır.
* result set sorgunun çıkış değerlerini tutar.
* resultset. next() metodu ile sıradaki satır alınır. Kayıtlar bittiyse "false" değer döner
* resultset, sıradaki satırın field'larını alırken getInt, getString, getDouble gibi metodlar kullanarak istediği formatta veriyi çekebilir.
* result set, server'daki buffer gibi kaynakları kullanır, bu yüzden diğer client'ların da server'a talepte bulunabilmesi için result set'le işimiz bitince kapatmalıyız.
* resultSetMetaData dönen bilgiye dair birtakım bilgileri (header gibi) içerir. Mesela, her field'ın ismi, tipi, boyutu, vs bu metadata'da tutulur.
* örneğin tablodaki field tiplerini bilmiyorsak (Ör: Dnumber int mi? Dname string mi bilmiyoruz varsayalım) getMetaData() fonk'u ile sırayla field'ların tipini öğrenebiliriz.

* **Önemli:** Eğer client tarafındaki uygulamayı parametreleştirmek istiyorsak; yani kullanıcıdan birtakım argumanlar alarak sorgumuzu yazacaksak preparedStatement class'ını kullanmalıyız.

* server'la işimiz bittiğinde bağlantıyı da koparmalıyız. Connection nesnesinin de close() metodu mevcuttur. Bunla bağlantıyı sonlandırırız.

* **Soru:** client tarafında metodun sonunda bir "finally" kısmı olmalı ve burda bağlantı koparılmalı. Neden?
* **Cevap:** Çünkü eğer client programı bir şekilde kesilirse (network'un kopması, elektrik kesilmesi, vs) ve bağlantı sonlanmazsa, bağlantı asla kendi kendine kapanamaz. Finally blok'u methodun düzgün çalışıp çalışmamasıyla ilgilenmeden muhakkak bulunduğu metodun sonunda çalışır. Bu bloğun içine conn.close() yaparak bağlantımızı her koşulda kapatabilmiş oluruz.

```
String query = "select LNAME,SALARY from EMPLOYEE where SSN = ?";
PreparedStatement p = conn.prepareStatement (query);
ssn="123456789";
p.clearParameters();
p.setString(1,ssn);
```

Eğer sorguda ikinci bir "?" işareti olsaydı. Bu parametrenin de mesela int tipinde olmasını isteseydik, bunu set etmek için şöyle demeliydik: `p.setInt(2, parameter_2);`

**Bu kavramlar ile ilgili diğer örnekler:**
Aşağıdaki sorguların sql ifadelerini yazmak için Java ortamından PostgreSQL sunucusuna bağlanılacaktır.

1. "DatabaseSystems" projesinde çalışan kadın işçilerin ad, soyad ve maaşlarını listeleyin.

2. "Chicago"da bulunan departman(lar)da çalışan işçilerin isim ve maaş bilgilerini yazdırın.

3. Her bir departmandaki çalışanların sayısını bulup departman ismine gore alfabetik olarak listeleyin.

4. "ProductX" projesinde kaç kişinin çalıştığını ve bu çalışanların ortalama maaşını bulun.

**Sorgular, Java'dan PostgreSQL sunucusuna <u>bağlanmadan</u> yazılacak olsaydı:**

1. select fname , lname from employee e, works_on w, project p
where w.essn=e.ssn and w.pno=p.pnumber and e.sex= 'F' and pname='DatabaseSystems'

2. select fname, lname, salary from employee e, dept_locations dl where
e.dno=dl.dnumber and dlocation = 'Chicago'

3. select dname, count(*) from department d, employee e where d.dnumber= e.dno group by dname order by dname

4. select count(*), avg(salary) from employee e, project p, works_on w where
e.ssn=w.essn and p.pnumber=w.pno and pname = 'ProductX'

Şeklinde olurlardı.

**********
**********

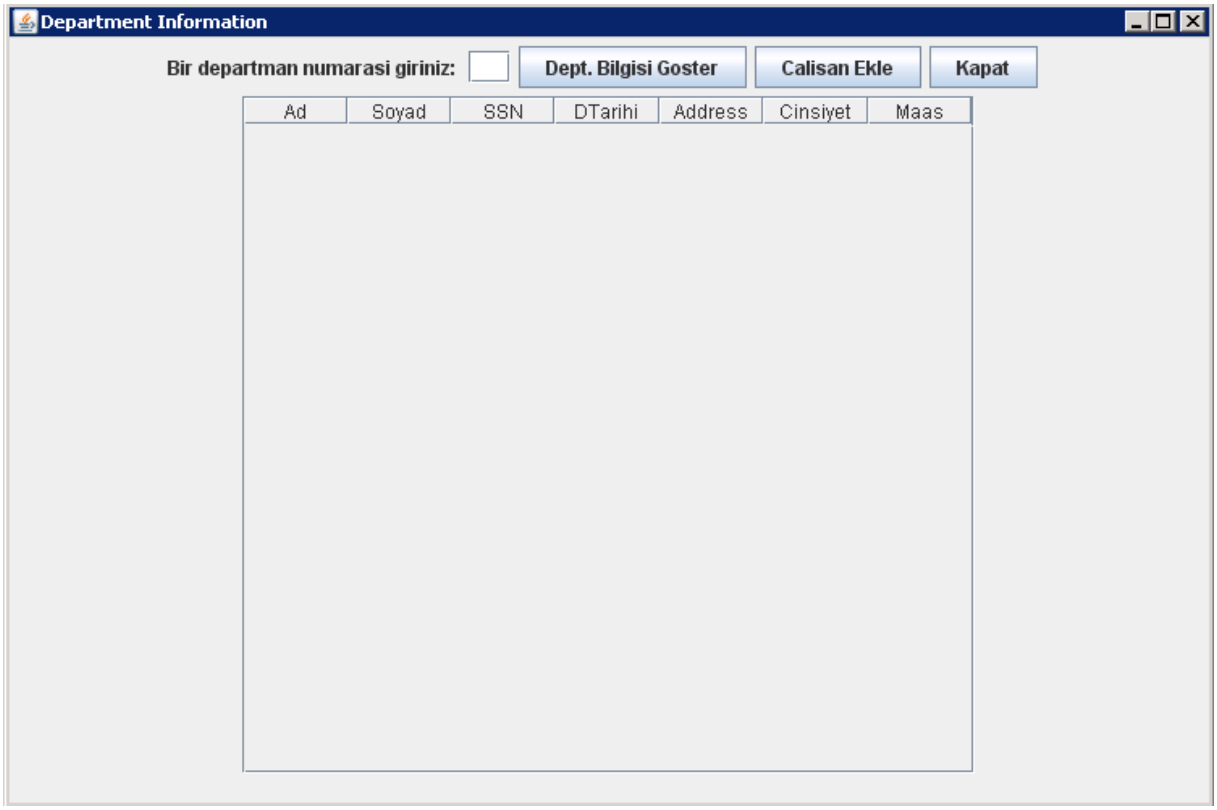
**Şimdi de bir frame yardımı ile basit bir arayüz oluşturulalım:**

Normalde JDBC ile ilgili (bağlantı kurma, sorgu yazma, bağlantı koparma, vs) kod kısımları ayrı class'larda yazılır böylece nesneye-dayalı kod yazma yapısı korunur ve esas uygulama ile araç olarak kullandığımız kısımlar karışmamış olur. Yukarıdaki örnekler kısa ve kolay yazılabilir olduğundan JDBC kodları ve uygulamalar iç içe aynı metotta yazılmıştır.

Aşağıdaki örnekte ise Java ile bir arayüz tasarlayıp, veritabanına bağlanarak; sorgulama ve DML işlemleri gerçekleştireceğiz. Bağlantı kurulumu ayrı bir class'ta; arayüze dair ayarlamalar ayrı bir class'ta; arayuzun komponentlerinin (buton, txt, vs) ne iş yapacağı ayrı bir class'ta yazılmıştır.

Ana metotun bulunduğu class, veritabanı bağlantısını yapar. Connection tipindeki bir nesneyi, Jframe tipindeki bir nesneye parametre olarak gönderir.

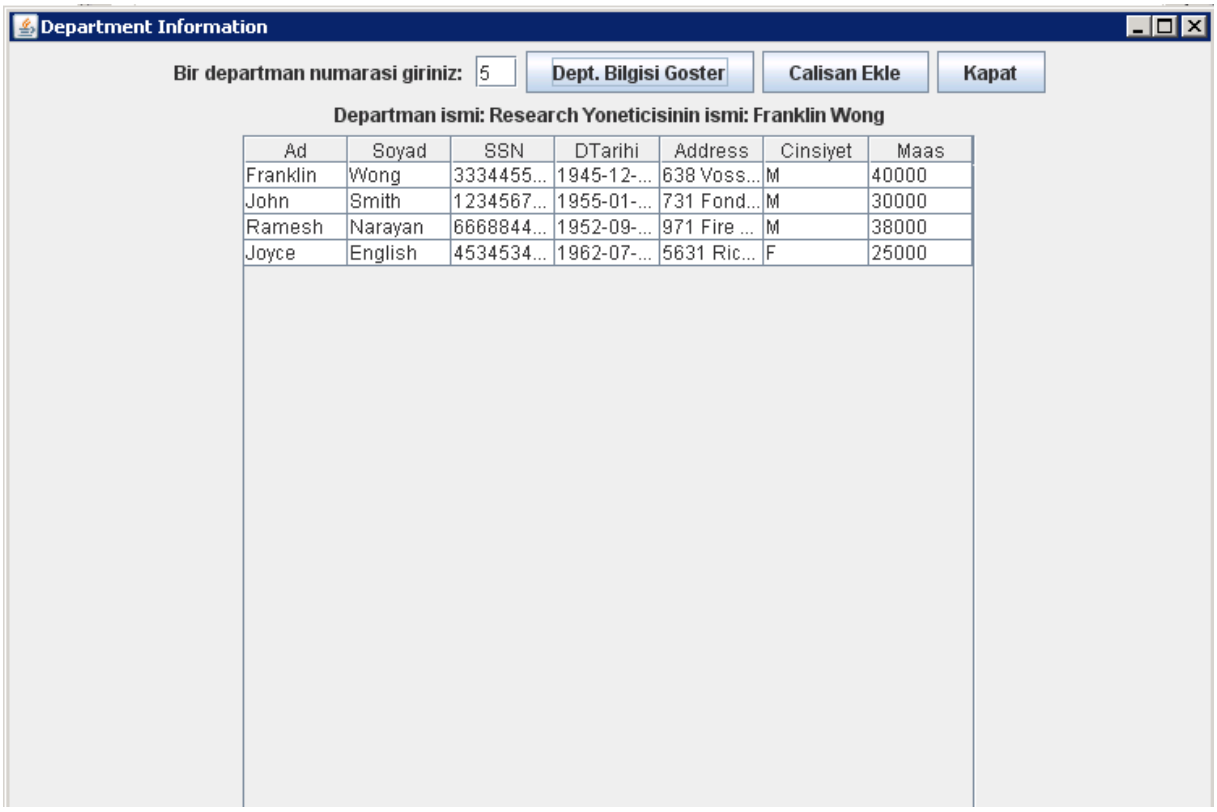
Jframe tipindeki nesne de arayüzümüzün boyutlarını belirler ve arayüzün üzerine koyacağımız Jpanel class'ının tipinde bir nesne yaratır. Connection nesnesini JPanel nesnesine parametre olarak verir.




Jpanel nesnesinde arayüzün komponentleri oluşturulur. Label, textbox. Butonlar, Jtable (employee bilgilerini gösteren data grid).

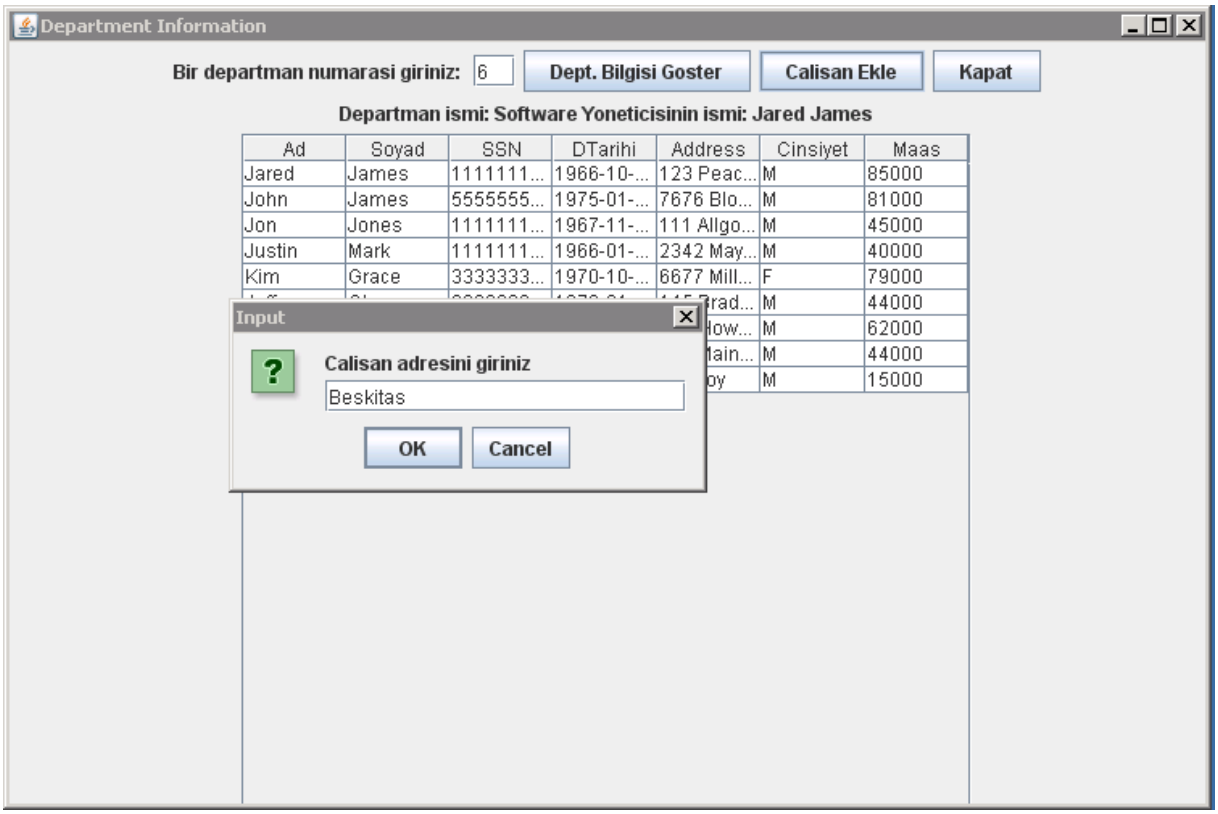
Textbox'a girilen departman numarasının, isim ve yönetici bilgilerini ve bu departmanda çalışan işçilerin bilgilerini görmek için "Dept. Bilgisi goster" butonu tıklanmalıdır.

Verilen bir departman no'suna göre ilk butona tıklandığında arayüz, aşağıdaki gibi olur:

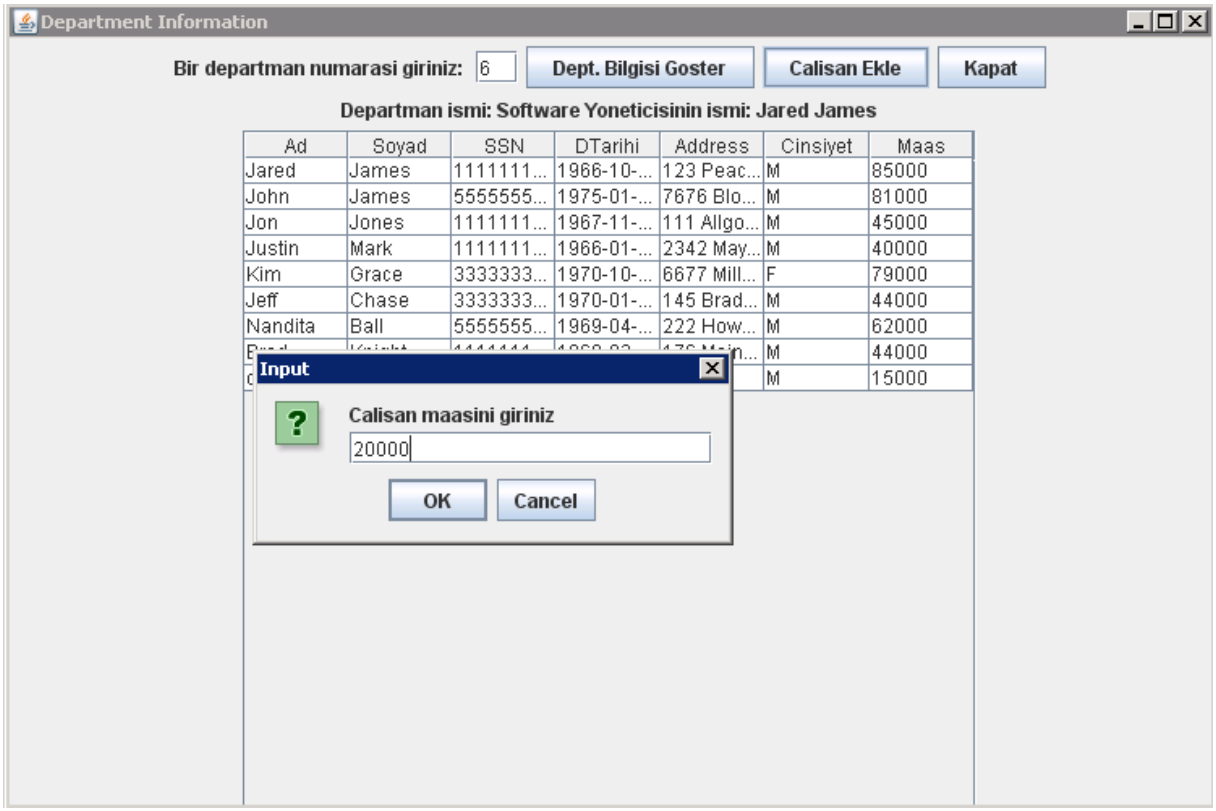



Employee tablosuna yeni bir çalışan eklemek için ikinci buton (Calisan Ekle butonu) tıklanmalıdır. Burda kullanıcıya inputMsjBox'lar gelecek ve kullanıcıdan birtakım bilgilerin

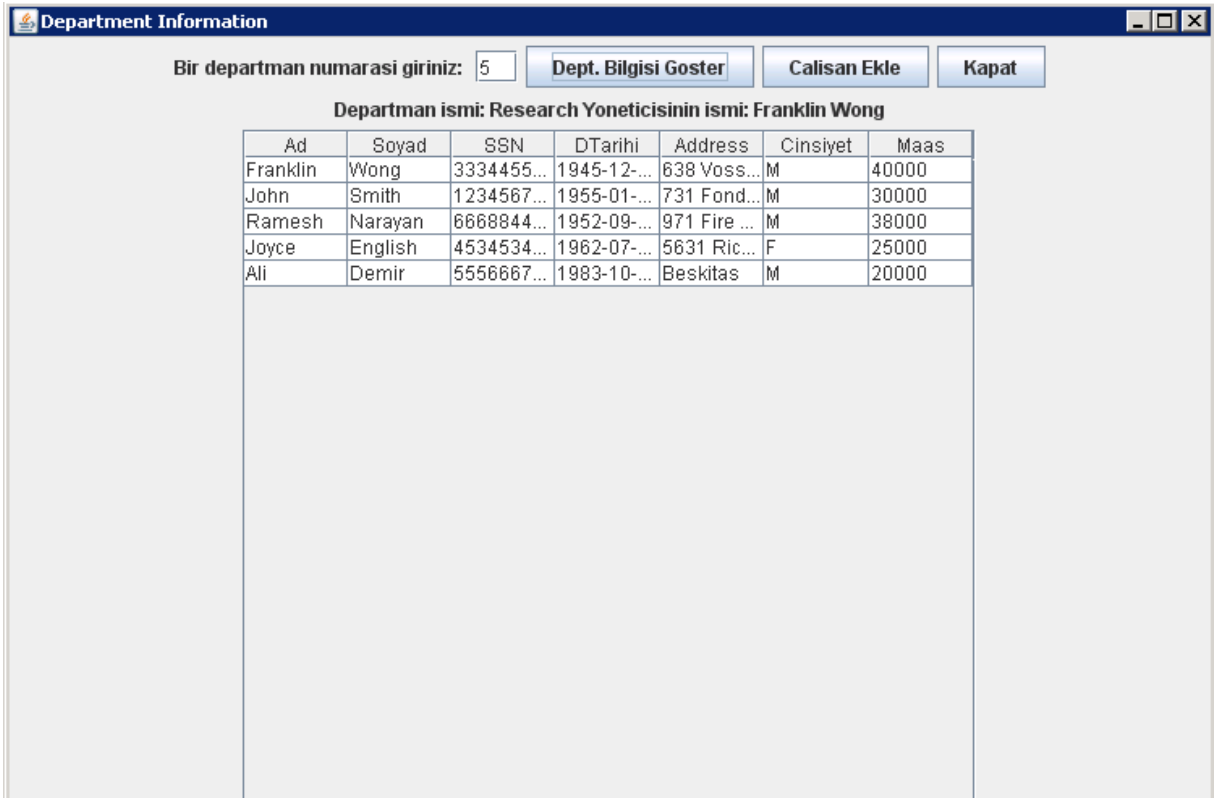
girilmesi istenecektir. (sırayla `fname,lname,ssn,bdate,address,sex,salary,dno bilgileri`). Örneğin çalışan adresini isteyen input mesajbox:




Maaş bilgisini isteyen input mesajbox:




5 no'lu departmana eklenen bu kişinin bilgilerini, bu departmandaki diğer çalışanlarla birlikte görmek için, departman numarası isteyen textbox'a 5'i girelim:




Üstteki arayüzde görüldüğü üzere: “Ali Demir” isimli yeni eklenen çalışanın bilgisi 5 no’lu departmanda çalışanların en altında listelenmektedir.

Arayüzün üçüncü butonu olan kapat’a tıklanınca, veritabanı bağlantısı koparılarak arayüz kapatılmaktadır: