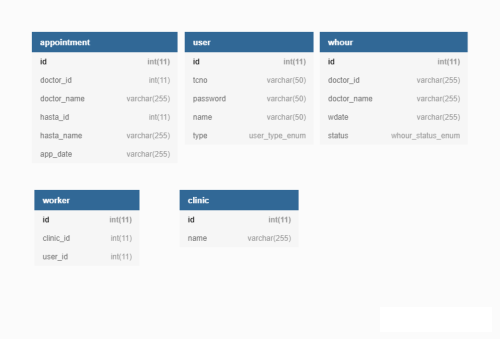
JAVA HASTANE OTOMASYONU

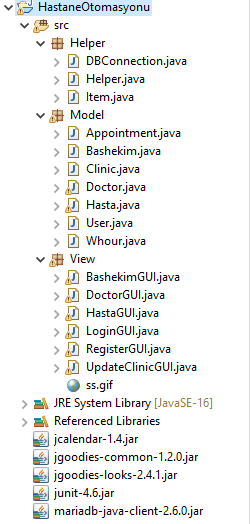
Eclipse üzerinden HastaneOtomasyonu isimli java projemizi oluşturduk, programı 3 temel yapıya böldük: Veritabanı bağlantısı ve kod tekrarına sebep olacak yapılar, veritabanı işlemleri, görsel arayüz. Böldüğümüz her bir yapı için ayrı ayrı paketler (package) oluşturduk.

1. Helper
2. Model
3. View

Bir otomasyon sistemi ürettiğimiz için veri giriş ve çıkışlarına uygun ‘hospital’ isimli bir veritabanı oluşturduk ardından aşağıdaki görseldeki şemada belirtildiği yapıda tablolar oluşturduk (koyu renkli yazılanlar primary key olarak seçilmiştir ve otomatik artış yöntemiyle (AUTO\_INCREMENT) veritabanına veri eklemektedir)

Veritabanına ait tabloların isimleri ve yapısı.



Helper paketinin yapısı veritabanı bağlantılarını sağlamak ve programın ilerleyen aşamalarında oluşacak hata mesajlarında kod tekrarına düşmeden önceden tanımlı mesajları gerekli sınıflara iletmek. Veritabanı bağlantısını sağlamak için kullanılan veritabanına uygun bir veri bağlayıcı gereklidir (database connector) biz veritabanı olarak kullanımı basit ve hızlı olduğu için Maria’yı seçtik ilgili veritabanının veri bağlayıcısını projemize ekledik (mariadb-java-client.jar).

Model paketinde ‘Helper’ paketinde ‘DBConnection’ sınıfında tanımladığımız veritabanı bağlantısını nesne olarak çağırarak veritabanı sorguları, ekleme, çıkarma, listeleme işlemlerini yapmak için kullanıyoruz. Aynı zamanda görsel arayüz (View paketi) kısmında ihtiyacımız olacak veritabanı işlemlerinin olduğu metotlar da bu paketin içerisinde sınıflara ayrılmış bir biçimde bulunmaktadır.

View paketini görsel arayüz için ayırdık, pencerelerin oluşturulması, dizaynı, veritabanına eklenecek verilerin girdisinin alınması ve gerekli yerlerde listelenmesi gibi işlemleri ‘Model’ sınıfında tanımlı metotların ilgili butonlar kullanıldığında çalışıp veritabanında işlem yapmasını bu paket üzerinden görsel olarak gösterebiliyoruz.

VERİ TABANINDA İŞLEMLER

Connection() sınfıyla database bağlantısı sağlarız. Connection c = DriverManager.getConnection(“bağlanılacak sql adresi”);

Veritabanındaki deki verileri alabilmemiz için bir connector kullanmamız gerek connector’ı bir kütüphane gibi düşünebiliriz. (Build Path > Configure Build Path > Libraries > Module Path > Add JARs) diğer kütüphaneleri de bu şekilde ekleyebiliriz.

**Statement Sınıfı**: Statement yapısına sql sorguları atayarak veritabanı işlemlerini java üzerinden yapabiliriz.

Statement st = c.createStatement(); şeklinde statement oluştururuz (c ifadesi connection dan ürettiğimiz nesneye karşılık gelmektedir)

String query = “INSERT INTO cars (cars\_name,cars\_speed) VALUES (‘Mercedes’,100)”;

Üstteki ‘query’ örneğinde sql sorgusu yazdık bu sql sorgusu database e veri eklemek için kullanılır(INSERT INTO) fakat sadece sorguyu yazmamız yetmez bunu ayrıyeten çalıştırmak içinde bir komuta ihtiyacımız var

st.executeUpdate(query); şeklinde girdiğimiz sorguyu çalıştırabiliriz (daha çok ekleme çıkarma sorguları için kullanılır).

Aynı zamanda : st.executeUpdate(“INSERT INTO cars (cars\_name,cars\_speed) VALUES (‘Mercedes’,100)”); şeklinde de query değişkenini kullanmadan çalışmasını istediğimiz sql komutunu çalıştırabiliriz. Benim query değişkenini kullanma sebebim karışık yapıları basite indirgemek ve hataları daha rahat görebilmek.

Statementlarla işimiz bittikten sonra bağlantıyı kapatmamız işleyiş ve güvenlik açısından son derece önemlidir.

st.close(); komutuyla üstte tanımladığım statement’ın bağlantısını kapatabiliriz.

**PreparedStatement:** Kullanıcıdan birden fazla bilgi alınırken kullanılır statement yapısına çok benzerdir.

String q = “INSERT INTO cars (cars\_name,cars\_speed) VALUES (?,?)”;

PreaparedStatement pt = c.prepareStatement(q);

Dikkat edileceği üzere statement dan farklı olarak atanacak değerler kısmına (VALUES) soru işaretleri geldi çünkü buraya gelecek veriler kullanıcıdan alınacak özellikle kullanıcıdan fazla veri almayı gerektiren işlemlerde PreparedStatement çok işlevseldir.

Soru işaretlerine değer atamak için :

pt.setString(1,name); (‘name’ daha önce tanımlanmış string bir değer alacak bir değişkendir)buradaki ilk değer yani 1 hangi soru işaretinin değiştirileceğini ifade eder ikinci değer ise “BMW” seçilen soru işaretine hangi değer atanacağını gösterir setString() metodunu kullanmamızın sebebi ise veritabanımızdaki ilk soru işaretinin değerinin string olması.

pt.setInt(2,speed); (‘speed’ daha önce tanımlanmış integer değer alacak bir değişkendir)

pt.executeUpdate(); daha önce de bahsettiğimiz executeUpdate() metodunu kullanarak yapılan sorguları ve işlemleri çalıştırıyoruz.

**ResultSet:** Veritabanından gelen bilgileri kullanmamıza yarar (bir sorgu sorduğumuzda geri dönen cevabı verir)

executeQuery() komutu executeUpdate() den farklı olarak işlemin yapılıp yapılmadığını değil sorgunun kendisini geriye döndürür (executeUpdate 0,1 döndürürken executeQuery sorgunun kendisini döndürür).

Statement st = c.createStatement();

ResultSet rs = st.executeQuery(“SELECT \* FROM cars”);

while(rs.next()){ rs.next() bir sonraki değeri var mı bakmamızı sağlar

System.out.println(rs.getString(“cars\_name”)+”-“+rs.getInt(“cars\_speed”));

}

bu komut dizisi ekrana ‘cars’ tablosundaki bütün verileri ekrana getirir.

Değerler ResultSet’lerin içine atanır atanan değerler de döngüler yardımıyla ekrana bastırılır.

GENEL YAPILAR

**Getter ve Setter :** Java’da başka bir classta tanımlanmış olan private değişkenlerin kullanılmasını sağlayarak data encapsulation durumunun sağlanmasına olanak tanıyan metodlara Setter ve Getter denir.

Get ile alıp ekrana çıktı çıkarma imkanımız oluyor(Çalıştırma). Set ile düzenleme hakkımız oluyor(değer verebiliyoruz).

Get bir return değeri dönderirken Set return değeri döndermez.

Data Encapsulation Kavramı: Verilerin dış ortamdan olası aksaklıklar karşısında koruma mekanizması olarak adlandırılır.

this Kavramı: Parametre olarak verilen ismin sınıf elemanıyla karışmaması için kullanılır.

**Metotların Ezilmesi (@Override):** Override kelimesinin anlamı ezmektir java’da kullanımı ise kelime anlamına benzerdir üst sınıftan alınan bir metodu alt sınıfta değiştirmektir. Başka bir deyişle kalıtım ile aldığımız metodu değiştirmek istersek o zaman override etmemiz gerekir.

**Override Tanımlama Kuralları :**

* Üst sınıftan override edilen metot alt sınıfta aynı isimle tanımlanmalıdır.
* Dönüş tipi aynı olmalıdır.
* Private,static,final metotları ile override yapılamaz. Ancak tekrar tanımlanarak kullanılabilirler.
* Metotlar, yalnızca alt sınıf (subclass) tarafından miras(extends) alınmışsa override işlemi uygulanabilir.
* Bir metot miras (extends) almıyorsa override işlemi uygulanamaz.

***Örnek:* Bir otomobil sınıfımız olsun onun altında da Mazda RX8 ve Opel Astra bulunsun ve otomobilde ortak olmayan özzellikleri override ederek değiştirelim.**

//Otomobil Sınıfı

public class Otomobil {  
    private int motorgucu;  
    private String renk;

    public Otomobil() {  
        // TODO Auto-generated constructor stub  
    }

    public Otomobil(int motorgucu, String renk) {  
        super();  
        this.motorgucu = motorgucu;  
        this.renk = renk;  
    }

    // Override edilmezse default olarak yakıt türü Benzinlidir.  
    public String yakitTuru() {  
        return “Benzin”;  
    }

    // Override edilmezse default olarak vitesTuru Otomatik.  
    public String vitesTuru() {  
        return “Otomatik”;  
    }

    // Override edilmezse default olarak kasaTipi Sedandır.  
    public String kasaTipi() {  
        return “Sedan”;  
    }

    // getter setter  
    public int getMotorgucu() {  
        return motorgucu;  
    }

    public void setMotorgucu(int motorgucu) {  
        this.motorgucu = motorgucu;  
    }

    public String getRenk() {  
        return renk;  
    }

    public void setRenk(String renk) {  
        this.renk = renk;  
    }

}

//MazdaRX8 sınıfı

public class MazdaRX8 extends Otomobil {

    private int km;

    public MazdaRX8() {  
        // TODO Auto-generated constructor stub  
    }

    public MazdaRX8(int km) {  
        super();  
        this.km = km;  
    }

    // Constructor Chaining  
    // super metodu ile  
    // Üst sınıf olan Otomobilin özelliklerinide belirttik  
    public MazdaRX8(int motorgucu, String renk, int km) {  
        super(motorgucu, renk);  
        this.km = km;  
    }

    // Sadece vites türünü override ettim.  
    // Diğerleri Aracta default olarak aynı  
    // Onun için override etmemize gerek yok  
    @Override  
    public String vitesTuru() {  
        return “Manuel”;  
    }

    // getter setter  
    public int getKm() {  
        return km;  
    }

    public void setKm(int km) {  
        this.km = km;  
    }

}

//OpelAstra sınıfı

public class OpelAstra extends Otomobil {  
    private int km;

    public OpelAstra() {  
        // TODO Auto-generated constructor stub  
    }

    public OpelAstra(int km) {  
        super();  
        this.km = km;  
    }

    public OpelAstra(int motorgucu, String renk, int km) {  
        super(motorgucu, renk);  
        this.km = km;  
    }

    // Otomobilin tüm özelliklerini Override Ediyoruz.  
    // Çünkü tüm özellikleri default tanımlanandan farklı.  
    @Override  
    public String kasaTipi() {  
        return “Hatchback”;  
    }

    @Override  
    public String yakitTuru() {  
        return “Dizel”;  
    }

    @Override  
    public String vitesTuru() {  
        return “Manuel”;  
    }

    // getter setter  
    public int getKm() {  
        return km;  
    }

    public void setKm(int km) {  
        this.km = km;  
    }

}

**ArrayList yapısı:** ArrayList, eleman eklendiği zaman büyüyen eleman çıkarıldığı zaman küçülen ve kendi içinde işlem yapmak için özel metot ve özelliklere sahip bir yapıdır. Veri yapısı olarak ifade edilir. Array ile ArrayList arasındaki farkı Array bir ip ArrayList’i bir lastik olarak düşünerek daha rahat kavrayabiliriz.

ArrayList<Type> listeAd = new ArrayList<Type>(); şeklinde tanımlanabilir

***Örnek:***

ArrayList<Integer> sayilar = new ArrayList<Integer>();

sayilar.add(12);

sayilar.add(5);

sayilar.add(45);

for (int sayi:sayilar){

System.out.println(sayi);

}

**Super():** Bir alt sınıf üst sınıfına erişmek isterse super() anahtar sözcüğünü kullanabilir. Bir alt sınıf super() metodunu kullanarak, üst sınıfının bir nesnesini yaratabilir ve onun değişkenlerine değer atayabilir.

***Örnek:***

class Kutu{

double en;

double boy;

double yukseklik;

Kutu (double e, double b, double y) {

en = e;

boy = b;

yukseklik = y;

}

}

/\* super() metodunun kullanılışı :

alt sınıfın super() metodunu kullanarak üst sınıfın bir nesnesini yaratabilir ve onun değişkenlerine değer atayabilir \*/

class AltKutu extends Kutu {

double agr; //kutunun ağırlığını tutacak değişken

/\* super() metodu ile en,boy ve yükseklik değerlerine atama yapıyor \*/

AltKutu (double e, double b, double y, double a) {

super(e,b,y);

agr = a;

}

}

**Miras(Extends):** Bir sınıftan yeni sınıflar türetmek olarak açıklayabiliriz örnek olarak yeni doğan bir çocuğun anne ve babasından bir takım özellikler taşıması olabilir

***Örnek:***

public class calisan {

int boy;

int kilo;

int yas;

}

class yonetici extends calisan {

String gorev;

}

‘yonetici’ isimli class’da miras(extends) kullanarak boy,kilo,yas değişkenleri de tanımlanmıştır fark olarak ekstradan ‘gorev’ değişkenini de içermektedir.

**Özel Hata Ayıklama(Exception):** Aslında Exception’lar kontrol edilen ve kontrol edilmeyen olmak üzere 2’ye ayrılır.

Çalışma zamanı sırasında meydana gelen bazı exceptionların kontrol edilmesi pogramcının isteğine bağlıdır. Bu tip exception’lar “RunTimeException” nesnesinden türemişlerdir. Programcı bu exceptionları ele almazsa program bir hata mesajı ile sonlanır. Ya da programcının ele aldığı şekilde akışına devam eder.

Kontrol edilen exceptionlar programcının isteğine bağlı değildir. Bu exceptionların ele alınması zorunludur. Mesela bir dosyanın açılması okunması ya da kapanması işlemleri kontrol edilen exceptionlardır. Bu tür exceptionlar programcının hatasından kaynaklanmayan exception’lardır (Bozuk bir dosya ya da URL)

NOT: Aslında bütün exception’lar Throwable nesnesinden türemiştir.

Exceptionlar üretildiği anda programcı tarafından ele alınmazsa program exception ile birlikte sonlanır. Bu nedenle bir exception ele alınmadan program akışına devam edilemez.

GÖRSEL ARAYÜZ (SWING KÜTÜPHANESİ)

**JTable:** Bize bir tablo verir, bu tabloya istersek komutlar (multidimensional array) kullanarak istediğimiz sayıda satır ve sütün oluşturup veri girebiliriz ve ya veritabanımızdan gelmesini istediğimiz veri tablosunun satır ve sütün sayısı ile birebir aynı oluşturduğumuz tabloya verileri de yazdırabiliriz.

**Absolute Layout:** JFrame ile oluşturduğumuz arayüze eklediğimiz eklentileri (butonlar,textboxlar) arayüzün her kısmına erişilebilir kılar bu şekilde eklentileri arayüzün bütün pixellerinde hareket ettirebiliriz.

**JPanel:** Kullandığımız nesneleri ayrı ayrı paneller içerisinde gruplayarak JFrame’e yani pencereye ekleyebiliriz, kod ve görsellik açısından düzen sağlar.