Pristupanje bazama podataka

JDBC

- JDBC (Java Database Connectivity) – biblioteka klasa koja omogućava povezivanje Java programa sa relacionim bazama podataka, (JDBC klase su deo java.sql)

- Klasa koja se naziva drajver deluje kao neka vrsta mosta između programa i izvora podataka, (za svaku popularnu bazu podataka postoji odgovarajući drajver).

- SQL (Structured Query Language) je standardni jezik kojim se pojednostavljuje korišćenje programa koji pristupaju relacionim bazama.

- Zahtev za očitavanjem zapisa iz baze podataka naziva se *upit*.

Drajveri baza podataka

- Neophodno je da koristite poseban drajver za svaki format baze podataka koji koristite u svom programu, a ponekad postoji i nekoliko različitih verzija drajvera za jedan isti format.
- Postoji poseban format za Java DB bazu podataka.
- Proizvođači MySQL baze podataka nude Connector/J, besplatan JDBC drajver sa otrovenim kodom.
- http://dev.mysql.com/downloads/connector/j/

Učitavanje drajvera

- Prvi zadatak u JDBC programu je učitavanje jednog ili više drajvera koji omogućavaju povezivanje sa izvornim kodom podataka.

  Drajveri se učitavaju korišćenjem metode:

  `Class.forName(String a)`

- Klasa Class, predstavlja deo java.lang paketa, i koristi se za učitavanje klasa u Java interpretator.

  Ovaj metod može da generiše izuzetak tipa:

  `ClassNotFoundException.`

- Nakon toga, konekciju sa izvorom podataka ostvarujemo korišćenjem DriverManager klase, koja se nalazi u java.SQL paketu.

- Metod **`getConnection(String a,String b, String c)`** klase DriverManager može se upotrebljavati za uspostavljanje konekcije. Ovaj metod vraća referencu na **`Connection`** objekat, koji predstavlja aktivnu konekciju!!!

  a – naziv koji identifikuje izvor podataka

  b – korisničko ime

  c – lozinka

- Metod `getConnection`, genreiše greške tipa SQLException

Očitavanje podataka korišćenjem SQL naredbi

- SQL naredba se u programskom jeziku Java predstavlja u obliku **Statement** objekta. Statement predstavlja interfejs.

- Objekat klase koja implementira interfejs dobija se kao rezultat izvršavanja **createStatement()** metoda Connection objekta.

- SQL naredbe se izvršavaju korišćenjem metode **`executeQuery (String a)`**

  `a` – treba da bude SQL upit koji je difinisan na osnovu sintakse SQL jezika.

SQL upit

- String s="SELECT Country, Year FROM Coal" + "WHERE (Country Is Not Null) ORDER BY Year"

  executeQuery(s)

- Ukoliko SQL upit ne sadrži sintaksne greške, executeQuery() će vratiti ResultSet objekat koji sadrži sve zapise sa određenim karakteristikama.

- Metode iz klase ResultSet za ekstrahovanje informacija iz zapisa:

  `getDate(String a), getDouble(String a), getFloat(String a), getInt(String a), getLong(String a), getString(String a),`

SQL upit

- Preuzimanje narednog zapisa se omogućava izvršavanjem `next()` metode ResultSet objekta.

- Nakon što se završi korišćenje konekcije, vrši se zatvaranje konekcije metodom `close()` bez argumenata.

Dodavanje zapisa u bazu

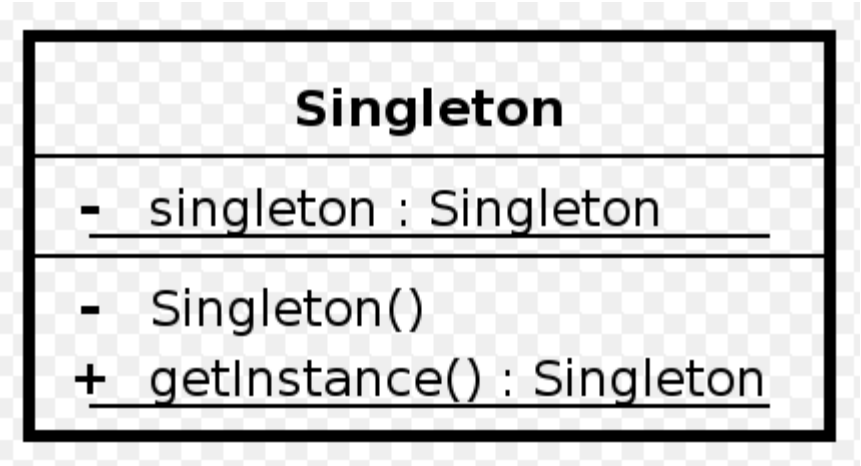
- Dodavanje zapisa se izvršava pomoću metode executeUpdate().

Primer za Java DB

```
try{

   Class.forName("org.apache.derby.jdbc.ClientDriver");
   Connection c = DriverManager.getConnection
   ("jdbc:derby://localhost:1527/baza","root", " ");
   Statement st=c.createStatement();
   ResultSet  r=st.executeQuery("select * from tabela");

        while(r.next()){
              r.getString("prezime")));

         }

} catch(... ) {}
```

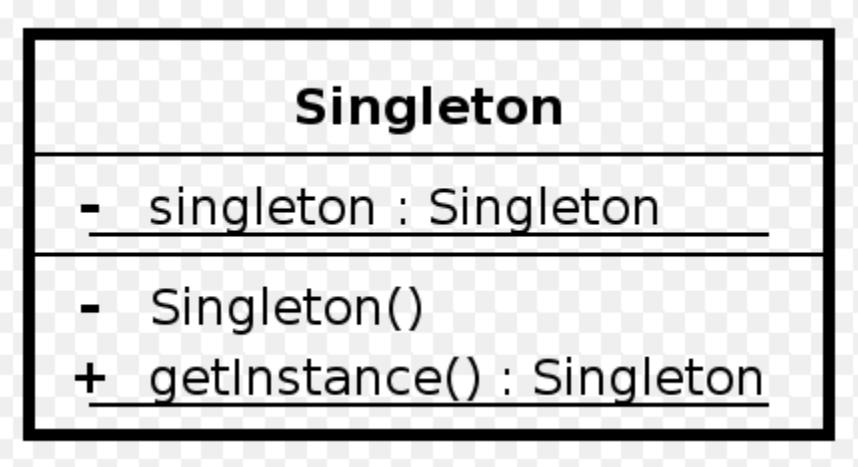
Singleton

- Objektni uzorak stvaranja
- Obezbeđuje da postoji samo jedan prikerak klase čije se stvaranje odlaže do prvog pristupa objektu
- Klijent dohvata unikatni objekat isključivo pristupnom metodom



Singleton

```
public class Singleton {

   private static final Singleton instance = null;

   private Singleton() {...}

   public static Singleton getInstance() {
        if(instance == null) instance = new Singleton();
        return instance; }

}
```



```
import java.sql.*;
public class Baza {
   private Connection c;
   private Statement st;
   private static Baza instanca;
   final String S1="jdbc:derby://loca...";

   public static Baza getInstanca() {
         if (instanca == null)
               instanca = new Baza();
         return instanca;
   }
   private Baza(){
        try {
         Class.forName("org.apache.derby.jdbc.ClientDriver");
        }
        catch (ClassNotFoundException ex)
        {...}
   }
```

```
public ResultSet Procitaj(String upit){
   ResultSet r = null;
   try {
        c=DriverManager.getConnection(S1,"root","");
        st = c.createStatement();
        try {
               r = st.executeQuery(upit);
        }
        catch (SQLException ex) {...}
   }
   catch (SQLException ex) {...}
    finally{
        try {
               c.close();
        }
        catch (SQLException ex) {...}
  }
  return r;
}
```

```
public int Zapisi(String zapis){
   int i = -1;
   try {
   c=DriverManager.getConnection(S1,"root","");
   st = c.createStatement();
   try {
                i = st.executeUpdate(zapis);
        }
        catch (SQLException ex) {...}
   }
   catch (SQLException ex) {...}
   finally{
        try {
                c.close();
        }
        catch (SQLException ex) {...}
  }
  return i;
}
}
```

```
import java.sql.Connection;
import java.sql.DriverManager;

public class DB {

    private static DB instance;
    private String kon = "jdbc:mysql://192.168.9.28:3306/mm?";
    private Connection con;

    private DB() {

        try {
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(kon, "java", "java123");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
public static DB getInstance() {
        if (instance == null) {
            instance = new DB();
        }
            return instance;
    }

    public synchronized Connection getConnection() {
        return con;
    }

}
```

```
stmt = con.createStatement();
try {
  stmt.executeUpdate("DROP TABLE IF EXISTS ...");
} catch (SQLException e) {
}
finally {
  stmt.close();
}
```