Rad sa bazama podataka na programskom jeziku Java

Osnovni koncepti

dr Slavica Kordić

Nikola Todorović

Marko Vještica

Miroslav Tomić

Gergelj Kiš

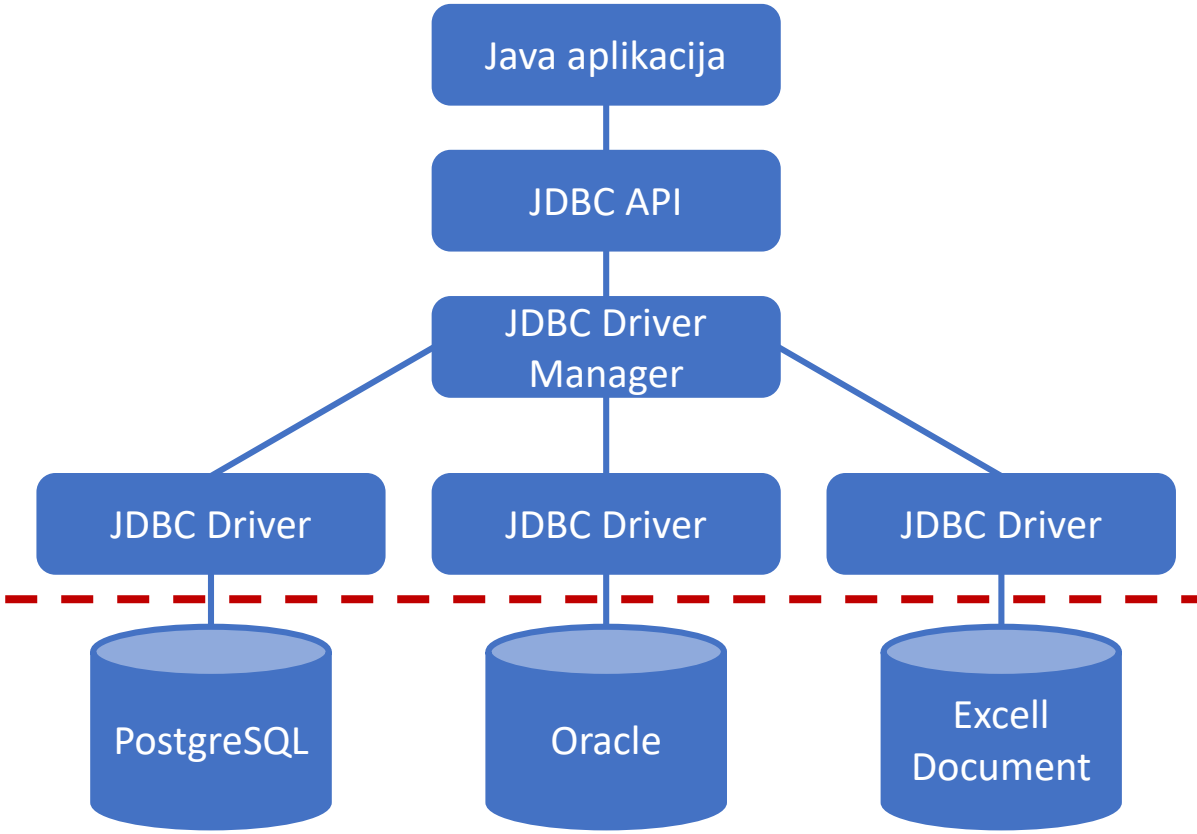
Aleksandar Hadžibabić

JDBC API

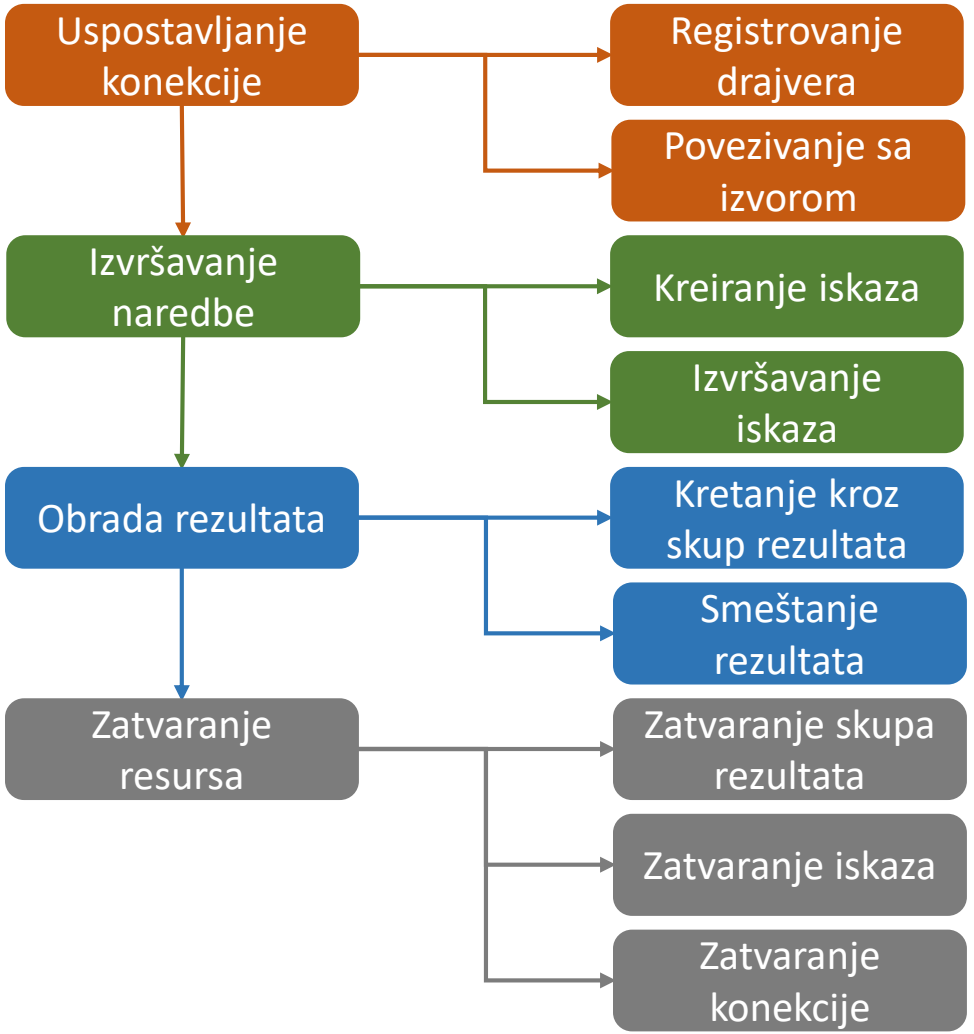
- Java Database Connectivity (JDBC) API
  - Java API (*application programming interface*) za povezivanje programa napisanih na Java programskom jeziku i velikog broja sistema za upravljanje bazama podataka i drugih vrsta izvora podataka.
  - Industrijski **standard** koji obezbeđuje povezivanje koje ne zavisi od korišćenog sistema za upravljanje bazama podataka (*database-independent*).
    - Definisan od strane *Sun Microsystems* (kreatori Java programskog jezika).
    - Dozvoljava proizvođačima SUBP-ova da implementiraju i naslede standard u okviru svojih implementacija **JDBC drajvera**.
  - Celokupan podsistem je definisan u standardnom paketu *java.sql*.
  - Zadaci JDBC-a:
    1. uspostavljanje konekcije sa bazom podataka,
    2. kreiranje i izvršavanje SQL naredbi,
    3. procesiranje rezultata.

JDBC - arhitektura

- **JDBC DriverManager** – komponenta koja upravlja drajverima za različite izvore podataka.
  - Osigurava da se za svaki izvor podataka koristi odgovarajući drajver.
- **JDBC Driver** – komponenta koja omogućava uspostavljanje konekcije sa različitim izvorima podataka.
  - Drajveri implementiraju protokol za prenos podataka između aplikacije i izvora podataka.
  - Za svaki zaseban izvor podataka mora postojati drajver.
- Ovakva arhitektura omogućava razvoj programa nezavisan od pozadinskog izvora podataka (***database-independent***).
  - U slučaju potrebe, moguće je samo zameniti izvor podataka, a da implementacija ostane ista (***hot swapping***).



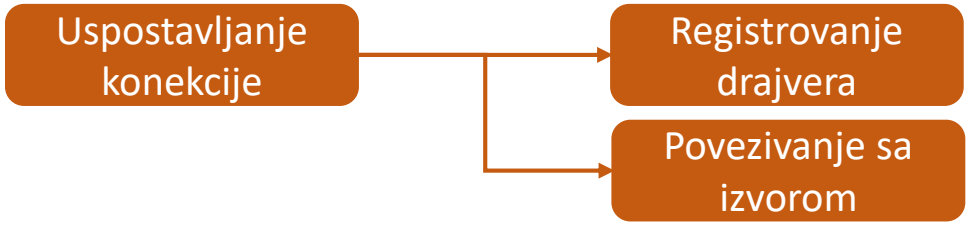
JDBC API – tok izvršavanja naredbe



JDBC - Elementi

- JDBC API elementi pomoću kojih se realizuje izvršavanje naredbi (klase i interfejsi):
  - *DriverManager*
  - *Connection*
  - *Statement*
  - *PreparedStatement*
  - *CallableStatement*
  - *ResultSet*
  - *ResultSetMetaData*
  - *DatabaseMetaData*
  - *SQLException*

JDBC API – uspostavljanje konekcije



JDBC API – uspostavljanje konekcije

- Pozivom sledeće metode kreiramo i registrujemo objekat specificirane drajver klase:

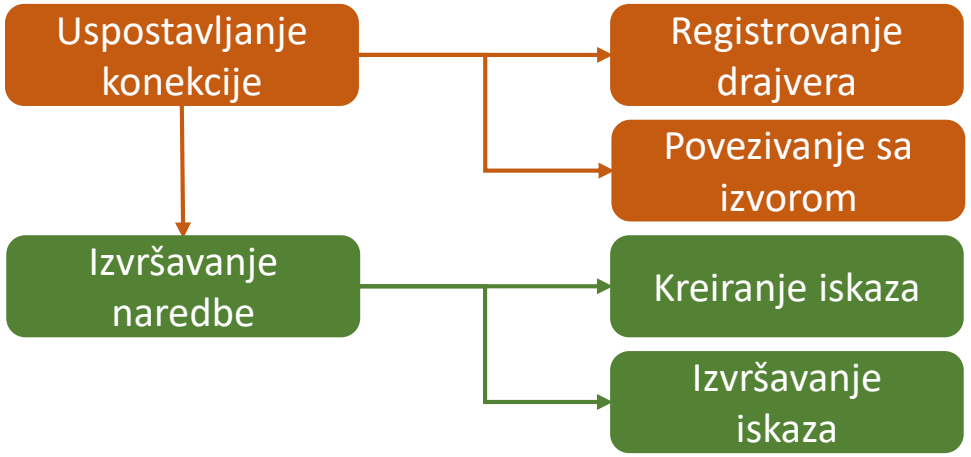
```
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
```

- Pozivom sledeće metode specificiramo *DriverManager-u* da pokuša da kreira konekciju na zadati URL, korišćenjem prethodno registrovanog drajvera:

```
Connection connection = DriverManager.getConnection(CONNECTION_STRING,
                                                    USERNAME,
                                                    PASSWORD);
```

  - Pomoću konekcionog stringa specificiramo koji drajver bi trebalo koristiti, kao i parametre konekcije za SUBP koji koristimo:
    - Konekcioni string za rad na lokalnoj mašini: ***"jdbc:oracle:thin:@localhost:1521:xe“***
      - Drajver koji se koristi, IP adresa, port i naziv baze sa kojom se povezujemo
    - Konekcioni string za rad u učionici: ***"jdbc:oracle:thin:@192.168.0.102:1522:db2016"***
  - **Uspostavljene konekcije predstavljaju resurs koji se na kraju rukovanja mora zatvoriti!**

JDBC API – izvršavanja naredbe



JDBC API – izvršavanje naredbe

- Izvršavanje naredbe se sastoji iz dva koraka:
  - kreiranje iskaza:
    - pomoću konekcije kreiraju se objekti klasa koje implementiraju sledeće interfejse:
      - *Statement* – koriti se za kreiranje osnovnih iskaza,
      - *PreparedStatement* – koristi se za kreiranje parametrizovanih iskaza,
      - *CallableStatement* – koristi se za kreiranje iskaza kojima se pozivaju uskladištene procedure i funkcije.
  - izvršavanje iskaza:
    - poziv odgovarajućeg oblika metode za izvršavanje:
      - *execute()*,
      - *executeQuery()*,
      - *executeUpdate()*.

JDBC API – interfejs *Statement*

- Osnovni interfejs za kreiranje i izvršavanje iskaza.
- Objekti klase koja implementira interfejs *Statement* kreiraju se pomoću `connection.createStatement()` metode.
- Može se koristiti za osnovne upite:
  - primeri - *Example01_Query* i *Example02_QueryElegant;*
  - iskazi za upite se izvršavaju pozivom `statement.executeQuery()` metode.
- Ne podržava direktnu parametrizaciju upita:
  - parametrizacija se može izvršiti konkatenacijom stringova;
  - primer – *Example03_QueryWithParams.*
- Može se koristiti i za DDL i DML naredbe:
  - primer – *Example04_DDL_DML_QL;*
  - DDL iskazi se izvršavaju pozivom `statement.execute()` metode;
  - DML iskazi se izvršavaju pozivom `statement.executeUpdate()` metode.

Zadatak za vežbu

- Koristeći JDBC API, napisati program koji će izlistati mbr, ime, prz radnika koji rade na projektu sa šifrom 10, a ne rade na projektu sa šifrom 30.
    - Kreirati novu konekciju koristeći interfejs *`DriverManager`*;
    - Za realizaciju upita koristiti interfejs *`Statement`*.

SQLInjection napad

- *SQLInjection* (*SQLi*) je vrsta injekcionog softverskog napada u okviru kog se maliciozan kod „injektuje“ u string SQL naredbe.
- Kada se ovakva SQL naredba prosledi SUBP-u na parsiranje i izvršavanje, može izazvati nepredviđeno izvršavanje naredbe sa potencijalno katastrofalnim posledicama.
  - Može izazvati curenje „osetljivih“ podataka, modifikaciju podataka, izvršavanje administratorskih naredbi nad bazom podataka, a u nekim slučajevima čak i izdavanje naredbi operativnom sistemu na kom je SUBP pokrenut.
- Jedan od najčešće izvođenih napada!
- Predstavlja zloupotrebu bezbednostih manjkavosti programa – neophodno je iz programa ukloniti bezbednosne propuste.

Interfejs *Statement* i *SQLi* napad

- Konkatenacija stringova radi parametrizacije upita predstavlja veliki potencijalni bezbednosni propust.
  - Podaci koje korisnik treba da unese mogu sadržati maliciozan kod!
  - Primer - *Example05_SQLInjection.*
- Da bi se ovakve situacije izbegle, neophodno je izvršiti „sanaciju“ korisničkog unosa – uklanjanje potencijalno opasnih karaktera iz korisnički unetog teksta.
  - Primer – uklanjanje svih „**‘**“ i „**;**“ karaktera iz korisnički unetog teksta.
- Interfejs *PreparedStatement* podržava automatsku sanitizaciju vrednosti parametara!
- **Kad god se vrši parametrizacija upita, koristiti interfejs *PreparedStatement!***

JDBC API – interfejs *PreparedStatement*

- Izvršavanje upita u okviru većine SUBP-ova podrazumeva prolazak kroz sledeće korake:
  1. parsiranje SQL upita,
  2. kompajliranje SQL upita,
  3. planiranje i optimizaciju načina za dobavljanje rezultata,
  4. izvršavanje optimizovanog upita i vraćanje rezultata.
- Korišćenje interfejsa *`Statement`* podrazumeva da se za svako izvršavanje prolazi kroz sva četiri koraka.
- Korišćenje interfejsa *`PreparedStatement`* podrazumeva jedan prolazak.
  - Kroz prva tri koraka – prilikom kreiranja objekta klase koja implementira interfejs *PreparedStatement*
  - **Svako izvršavanje naredbe se svodi na izvršavanje 4. koraka**, uz opciono prosleđivanje vrednosti za parametre.
  - Značajno efikasnije prilikom ponovnog korišćenja istog objekta – **vrlo pogodno za parametrizaciju**.

JDBC API – interfejs *PreparedStatement*

- Specificiranje parametara:
  - mesto u upitu na koje bi trebalo da dođe konkretna vrednost označava se karakterom „**?**“.
- Postavljanje vrednosti parametara:
  - za postavljanje konkretne vrednosti na mesto parametra koriste se metode oblika *`setX(int parameterIndex, X value)`*:
    - *parameterIndex* – indeks parametra u upitu:
      - zavisi od redosleda karaktera „**?**“ u upitu;
      - **indeksiranje kreće od 1**;
    - *X* – predstavlja tip podatka za koji se postavlja vrednost;
    - *value* – vrednost koja se postavlja za parametar.
  - Često korišćene metode:
    - *`setString`*`(...)`,
    - *`setInt`*`(...)`,
    - *`setFloat`*`(...)`,
    - *`setDate`*`(...)`,
    - *`setBoolean`*`(...)`,
    - *`setNull`*`(...)`...

JDBC API – interfejs *PreparedStatement*

- Objekti klase koja implementira interfejs *`PreparedStatement`* kreiraju se pomoću *`connection.prepareStatement(String statement)`* metode.
- Može se koristiti za osnovne upite:
  - primer: - *Example01_Query;*
  - iskazi za upite se izvršavaju pozivom *`.executeQuery()`* metode.
- Parametrizacija je **direktno podržana**:
  - neophodno izvršiti specificiranje parametara prilikom kreiranja naredbe;
  - neophodno izvršiti postavljanje vrednosti parametara pre izvršavanja naredbe;
  - primeri – *Example02_QueryWithParams, Example03_SQLInjection.*
- Može se koristiti i za DDL i DML naredbe:
  - primer – *Example04_DDL_DML_QL;*
  - DDL iskazi se izvršavaju pozivom *`.execute()`* metode;
  - DML iskazi se izvršavaju pozivom *`.executeUpdate()`* metode.

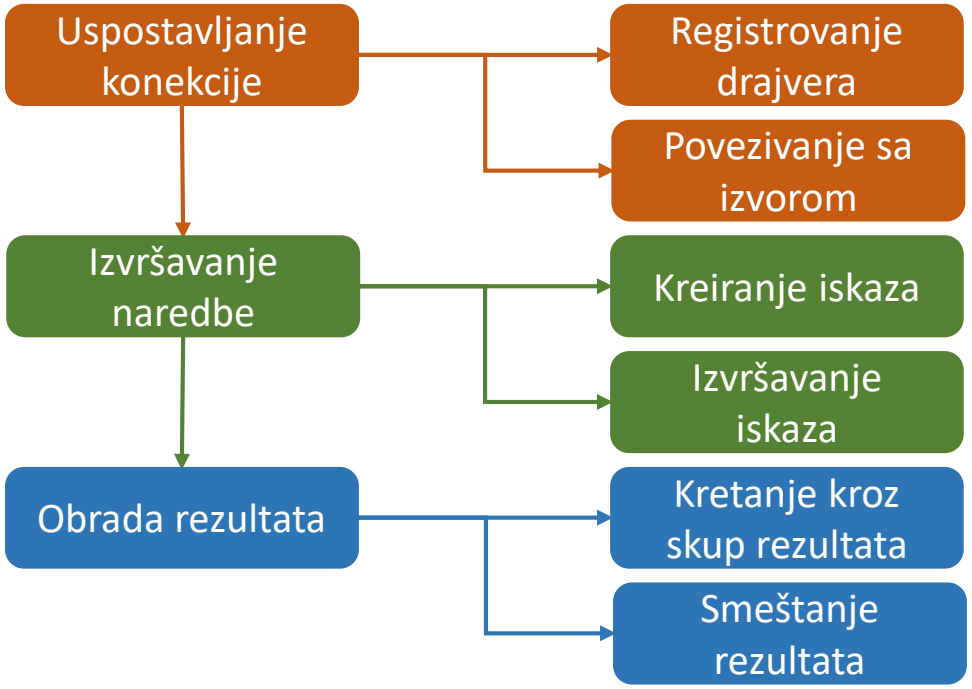
Zadatak za vežbu

- Koristeći JDBC API, napisati program koji će za unete spr i brc vrednosti izlistati broj radnika koji na zadatom projektu rade više od zadatog broja sati.
    - Za realizaciju upita koristiti interfejs *`PreparedStatement`*.

JDBC API – *Callable statement*

- Objekti klase koja implementira interfejs *`CallableStatement`* koriste se za izvršavanje uskladištenih procedura i funkcija.
- Kreiraju se pomoću *`connection.prepareCall(String statement)`* metode.
  - Primer: Example01_ExecuteFunction
    - Pre izvršavanja neophodno kreirati odgovarajuću uskladištenu funkciju:
      - *Example01_SelectProjekatFunction.sql;*
    - iskazi za upite se izvršavaju pozivom *`.execute()`* metode, dok se vrednosti izlaznih parametara preuzimaju pozivom *`.getObject(int parameterIndex)`*.
- Registrovanje izlaznih parametara:
  - Za registrovanje izlaznih parametara, preko kojih će biti vraćena vrednost iz procedure ili funkcije, koriste se metode oblika:
    - *`registerOutParameter(int parameterIndex, int sqlType)`*:
      - *parameterIndex* – indeks parametra u upitu:
        - zavisi od redosleda karaktera „**?**“ u upitu;
        - indeksiranje kreće od 1;
      - *sqlType* – predstavlja tip parametra.

JDBC API – obrada rezultata



Kursori

- Kursori predstavljaju kontrolnu strukturu pomoću koje je moguće vršiti prolazak kroz skup rezultata (***result set***) nekog upita po principu red-po-red.
  - Umesto da se ceo upit izvrši odjednom, kreira se kursor koji omogućava postupno čitanje rezultata upita.
- Jedan od razloga za ovakvo rukovanje podacima jeste izbegavanje situacije u kojoj je odjednom potrebno obraditi veliku količinu memorije, što može izazvati nedostatak memorije za obradu podataka (***memory overrun***).
- Predstavljaju koncept vrlo sličan konceptu iteratora u različitim programskim jezicima.
  - Kursor pokazuje na red koji je trenutno na redu za obradu.

JDBC API – interfejs *ResultSet*

- Koristi se za rukovanje pozadinskim kursorskim kontrolnim strukturama – kako bi se izvršila obrada skupa rezultata dobijenog izvršavanjem zadatog upita.
- Objekti klase koja implementira interfejs *ResultSet* kreiraju se pozivom *.executeQuery()* naredbe nad objektima iskaza.
- Primer iteriranja kroz rezultat upita - *Example01_Iterating*
  - Neposredno nakon kreiranja objekta, kursor pokazuje na poziciju pre prvog reda (*beforeFirst*).
  - Tip se definiše prilikom kreiranja objekta iskaza, prosleđivanjem odgovarajuće konstante. Tipovi kursora:
    - TYPE_FORWARD_ONLY – koriste se samo za kretanje u napred, podrazumevani tip;
    - TYPE_SCROLL_INSENSITIVE – *scrollable* kursor koji nije osetljiv na promene u originalnom skupu rezultata;
    - TYPE_SCROLL_SENSITIVE – *scrollable* kursor koji je osetljiv na promene u originalnom skupu rezultata.

Zadatak za vežbu

- Za svakog šefa ispisati ime, prezime, broj radnika kojima je direktno nadređeni i platu. Takođe, ispod svakog šefa dati prikaz svih radnika (ime, prezime i plata) kojima je direktno nadređeni.
  - Primer izlaza programa:

```
Savo Oroz,  2,  40000din
    Pero Peric, 30000din
    Eva  Ras,   40000din
...
```

JDBC API – interfejs *ResultSet*

- Preuzimanje vrednosti atributa:
  - Za preuzimanje vrednosti atributa trenutno obrađivane torke koriste se metode oblika *.getX(int columnIndex)*.
    - *X* – predstavlja tip podatka za koji se postavlja vrednost;
    - columnIndex – redni broj kolone rezultujućeg upita; indeksiranje počinje od 1.
  - Često korišćene metode:
    - `getString(...)`,
    - `getInt(...)`,
    - `getFloat(...)`,
    - `getDate(...)`,
    - `getBoolean(...)...`
  - Preuzete vrednosti se dalje mogu dodeliti nativnim java tipovima, objektima korisnički kreiranih klasa...

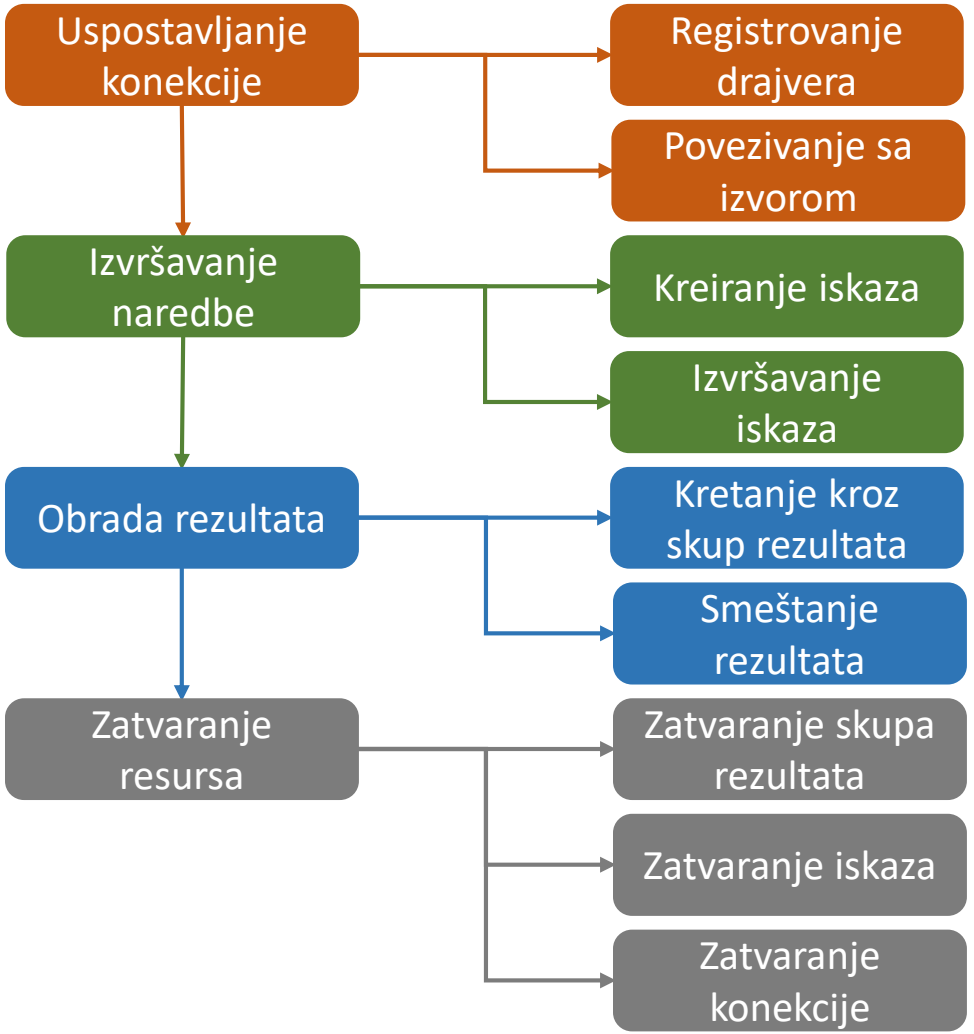
JDBC API – interfejs *ResultSet*

- Interfejs ResultSet može se koristiti i za ažuriranje podataka prilikom prolaska kroz rezultujuči skup podataka.
  - Definiše se prilikom kreiranja objekta iskaza, prosleđivanjem konstante CONCUR_*UPDATABLE*.
    - Može se definisati i da ažuriranje nije dozvoljeno, korišćenjem konstante *CONCUR_READ_ONLY* – podrazumevano ponašanje.
- Primer ažuriranja podataka – *Example02_Updating*
  - Unos:
    - podrazumeva pozicioniranje na specijalni red pod nazivom *InsertRow;*
    - potom se vrši postavljanje vrednosti za kolone nove torke korišćenjem metoda oblika `updateX(int columnIndex, X value);`
    - kada se završi postavljanje vrednosti, unos se vrši pozivom `insertRow()` metode;
    - **uneta torka NEĆE biti vidljiva u originalnom skupu rezultata!**
  - Izmena
    - podrazumeva pozicioniranje na žejeni red;
    - potom se vrši postavljanje novih vrednosti za kolone torke korišćenjem metoda oblika `updateX(int columnIndex, X value);`
    - kada se završi postavljanje vrednosti, unos se vrši pozivom `updateRow()` metode;
    - **unete izmene će biti vidljive u originalnom skupu rezultata.**
  - Brisanje
    - podrazumeva pozicioniranje na željeni red;
    - pa potom poziv metode `deleteRow();`
    - **torka će biti obrisana i iz originalnog skupa torki.**

Zadatak za vežbu

- U tabelu radnik dodati novog radnika, a potom ga angažovati na projektu sa najmanjom vrednošću obeležja spr.
    - Za realizaciju unosa koristiti interfejs *`ResultSet`.*

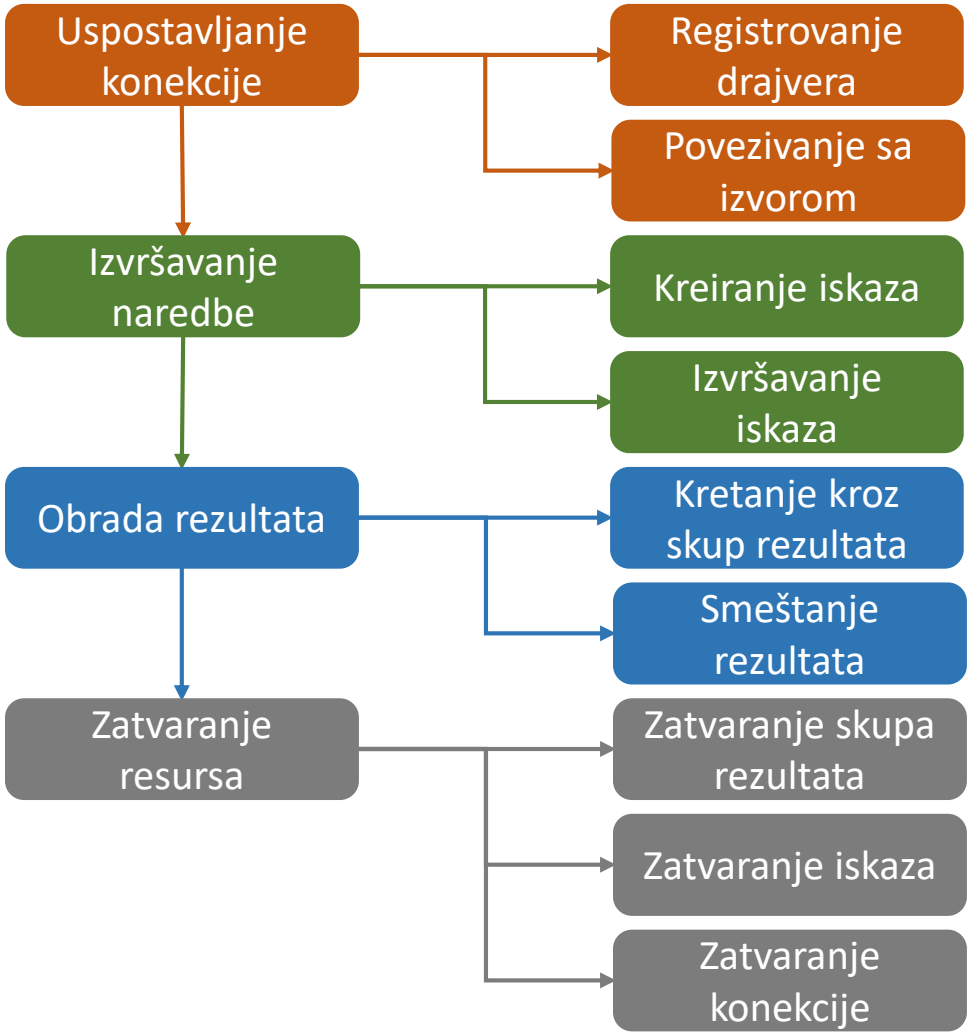
JDBC API – zatvaranje resursa



JDBC API – zatvaranje resursa

- Resursi koji se koriste prilikom rada sa različitim izvorima podataka obavezno se moraju eksplicitno zatvoriti.
  - Da bi se oslobodili zauzeti resursi – kursorska područja, *socket*-i za konekciju itd.
  - Na ovaj način se izbegava „curenje“ memorije (eng. *memory leak*).
- Resursi koji se moraju zatvoriti: `Connection`, `Statement`, `PreparedStatement`, `CallableStatement` i `ResultSet`.
- Zatvaranje resursa ne može se prepustiti *garbadge collector*-u!
  - Nije u mogućnosti da zatvori kursorsko područje u okviru SUBP-a.
- Za eksplicitno zatvaranje resursa koristiti:
  - *try-with-resource* iskaz za automatsko zatvaranje resursa, ili
  - eksplicitni poziv `.close()` metode nad resursom,
    - najčešće u sklopu *finally* bloka običnog *try-catch* bloka.

JDBC API – zatvaranje resursa



Napredne opcije

JDBC API – transakcije

- JDBC API pruža mogućnost upravljanja transakcijama korišćenjem objekata konekcije.
- Podrazumevani režim rada je *auto commit*, u kom se svaka izvršena naredba tretira kao zasebna transakcija – nakon svake uspešno izvršene naredbe pozvaće se naredba *commit*.
  - Ovakvo ponašanje može biti problematično u situaciji kada se vrši kompleksnija obrada podataka – integritet podataka može biti doveden u pitanje.
  - Primer - *Example01_AutoCommit*.
- Kako bi se ručno upravljalo transakcijama, neophodno je isključiti *auto commit* režim pozivom metode `setAutoCommit(false)` nad objektom konekcije koji koristimo za kreiranje iskaza.
  - Potom je moguće u odgovarajućem trenutku pozvati *commit* ili *rollback* naredbu, takođe korišćenjem objekta konekcije.
  - Primer - *Example02_ManualCommit.*

JDBC API – meta-podaci

- U nekim slučajevima može biti neophodno da se pristupi podacima o samoj bazi podataka i o tabelama u njoj – rečniku podataka (*data dictionary*).
- U tu svrhu postoje dve interfejsa:
  - `DatabaseMetaData`,
  - `ResultSetMetaData`.
- Primeri upotrebe:
  - `DatabaseMetaData.getTables()`,
  - `ResultSetMetaData.getColumnName()`,
  - `ResultSetMetaData.getColumnTypeName()`.

Dobre prakse

Connection Pooling

- Svako uspostavljanje veze sa izvorom podataka zahteva zauzimanje resursa za održavanje konekcije, kao i inicijalnu komunikaciju sa izvorom podataka.
- U aplikacijama koje intenzivno koriste usluge izvora podataka, ovo može imati značajan uticaj na performanse, jer se postupak kreiranja stalno iznova ponavlja.
- Zbog toga je uveden *Connection Pooling* koncept, koji predviđa keširanje određenog broja unapred pripremljenih konekcija.
  - Kad god je potrebna konekcija radi komunikacije sa izvorom podataka, preuzmemo je iz unapred pripremljenog „bazena“ konekcija.
  - Kada se završi rad sa konekcijom, ona se vraća u „bazen“ pozivom *.close()* metode.
- Jedna od najkorišćenijih Java implementacija koncepta – *HikariCP*.
  - Primer - *ConnectionUtil_HikariCP* i *Example01_ConnectionPool*.

DAO šablon

- *Data Access Object* (*DAO*) šablon je strukturalni šablon koji omogućava razdvajanje sloja poslovne logike (*bussines layer*) od sloja za perzistenciju podataka (*persistence layer*).
- Ovakvim razdvajanjem slojeva održava se princip jedinstvenog zaduženja svake komponente sistema (*Single Responsibility*)
  - Što je u skladu sa *SOLID* dizajn principima softvera, čiji je cilj da softversko rešenje bude razumljivo, fleksibilno i održivo.
- DAO sloj implementira logiku nad konkretnim entitetima – sadrži metode poput *`getX`*, *`insert`*, *`update`*, *`delete`*...
  - Za svaki entitet postojaće i zasebna klasa u DAO sloju.
  - Ostatak aplikacije komunicira sa izvorom podataka isključivo preko metoda koje pruža ovaj sloj.

Kraj prezentacije