OPRPP 2 - 2022./2023. - ZI

===== TEORIJA =====

1. Pretpostavimo da u bazi podataka testdb imamo tablicu autori s atributima *ime* i *brojKnjiga*, a upravitelj baze je Apache Derby.

a) Napišite najjednostavniji program koji će koristeći JDBC dohvatiti i ispisati imena i broj knjiga svih autora koji imaju barem 4 knjige. Napomena: nisu bitni do u slovo pogođeni nazivi metoda; ne treba obrađivati iznimke.

```
import java.sql.*;

public class AuthorBooks {
    public static void main(String[] args) {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Povezivanje na bazu
            con = DriverManager.getConnection("jdbc:derby://localhost/testdb");
            // Kreiranje SQL izraza
            stmt = con.createStatement();
            // Izvršavanje SQL upita
            rs = stmt.executeQuery("SELECT ime, brojKnjiga FROM autori WHERE brojKnjiga >= 4");
            // Obrada rezultata upita
            while (rs.next()) {
                String ime = rs.getString("ime");
                int brojKnjiga = rs.getInt("brojKnjiga");
                System.out.println("Ime autora: " + ime + ", Broj knjiga: " + brojKnjiga);
            }
        } catch (SQLException e) {
            // obrada iznimaka nije potrebna prema uputama
        } finally {
            // zatvaranje veze s bazom
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                // obrada iznimaka nije potrebna prema uputama
            }
        }
    }
}
```

b) Detaljno objasnite što se događa tijekom izvođenja tog programa od njegova početka pa trenutka do kada je stvoren upit (tko koga zove, što se događa i kakve veze ima informacija da je upravitelj baze Apache Derby).

Program započinje uvozom potrebnih klasa za rad s JDBC API-jem. Najvažnije su Connection, Statement i ResultSet klase koje omogućavaju povezivanje s bazom podataka, izvršavanje SQL upita i pristup rezultatima.

U main funkciji, prvo se kreiraju varijable za spajanje na bazu (Connection), izvršavanje upita (Statement) i skladištenje rezultata (ResultSet).

Sljedeći korak je stvaranje veze s bazom podataka. Za to se koristi metoda `DriverManager.getConnection()`, koja prima JDBC URL koji specificira bazu na koju se program treba povezati. U ovom slučaju, JDBC URL je "jdbc:derby://localhost/testdb", što znači da program treba povezati na Apache Derby bazu podataka pod nazivom "testdb" koja se nalazi na lokalnom stroju (localhost).

S obzirom na to da je upravitelj baze Apache Derby, moramo koristiti Derby JDBC driver. Java ima koncept driver managera, koji omogućuje programima da se povežu na bazu podataka koristeći JDBC driver. U ovom slučaju, kada se metoda `DriverManager.getConnection()` poziva, Java VM će automatski učitati i registrirati Derby JDBC driver.

Nakon što je veza s bazom podataka uspostavljena, program stvara SQL upit. Ovo se postiže pozivanjem metode `createStatement()` na objektu Connection, koji vraća novi Statement objekt. Na tom objektu se može pozvati `executeQuery()`, koja izvršava SQL upit na bazi podataka.

`executeQuery()` metoda prima string koji predstavlja SQL upit i izvršava ga na bazi podataka. Rezultat ovog upita (tj., sve redove koje upit vraća) se tada pohranjuju u ResultSet objektu.

U ovom slučaju, SQL upit je "SELECT ime, brojKnjiga FROM autori WHERE brojKnjiga >= 4". Ovaj upit dohvaća imena i brojeve knjiga svih autora iz tablice "autori" koji imaju barem 4 knjige.

Nakon što je upit izvršen i rezultati su pohranjeni u ResultSetu, program je spreman za obradu rezultata.

2. Razmatramo naprednije mogućnosti koje nudi JDBC.

a) Objasnite na koji standardizirani način JDBC omogućava klijentskom kodu komunikaciju s bazom podataka, ako klijent ne smije biti svjestan koju bazu podataka koristi.

JDBC API koristi JDBC URL za identifikaciju baze podataka s kojom se povezuje. Ovaj URL se sastoji od prefiksa "jdbc:", podtipa koji označava vrstu drivera, i podatka specifičnog za bazu koji identificira bazu podataka.

Kada Java program treba povezati se s bazom podataka, poziva se metoda DriverManager.getConnection() s JDBC URL-om baze podataka. DriverManager koristi ovaj URL da pronađe odgovarajući driver i uspostavi vezu.

Nadalje, JDBC API pruža standardizirani set metoda za izvršavanje SQL upita i obradu rezultata. Ove metode su iste za sve baze podataka, omogućujući klijentu da komunicira s bazom bez potrebe za poznavanjem specifičnosti baze podataka.

b) Objasnite kako se pod JDBC-om vise naredbi može izvesti pod jednom transakcijom.

1. postaviti autocommit na false: con.setAutoCommit(false);
2. zadati sve željene naredbe
3. pozvati con.commit() (ako želimo završiti transakciju) ili con.rollback() (ako odustajemo)

c) Objasnite čemu služe i na koji se način izvode bazeni veza. Prikažite detaljnijim pseudokodom implementaciju jednog bazena veza.

Koncept Bazena Veza funkcionira na način da sadrži otvorene veze koje onda „iznajmljuje“ korisniku, a ona se nakon korištenja vraća u bazen. Potreba za njima javlja se jer inače pri svakom pristupu bazi treba uspostavljati vezu što je vremenski skupo. Ovako je određen broj veza zadržan u bazenu i kad je potrebno pristupati bazi koristi se jedna od tih veza (*getConnection* iznajmljuje vezu a *close* ju vraća u pool).

```
DataSource pool = stvoriNekakoConnectionPool();
Connection c1 = pool.getConnection();
try(PreparedStatement p = c1.prepareStatement(...)) {...}
c1.close();
```

```
Connection c2 = pool.getConnection();
try(PreparedStatement p = c2.prepareStatement(...)) {...}
c2.close();
```

3. Za bazu podataka iz zadatka 1 napišite naredbeno-retčanu aplikaciju koja obavlja isti posao koristeći tehnologiju JPA. Prikažite sve relevantne razrede. Nije potrebno pisati sadržaj datoteke persistence.xml (pretpostavite da ste je korektno napisali i da ona postoji). Napomena: nisu bitni do u slovo pogođeni nazivi metoda; ne treba obrađivati iznimke.

```
@Entity
@Table(name = "autori")
public class Autor {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;

	@Column(name = "ime")
	private String ime;

	@Column(name = "brojKnjiga")
	private int brojKnjiga;

	// getteri i setteri...
}

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MyPersistenceUnit");
	EntityManager em = emf.createEntityManager();
	TypedQuery<Autor> query = em.createQuery("SELECT a FROM Autor a WHERE a.brojKnjiga >= 4",
Autor.class);
	List<Autor> autori = query.getResultList();
	for (Autor autor : autori) {
		System.out.println("Ime autora: " + autor.getIme() + ", Broj knjiga: " +
	autor.getBrojKnjiga());
	}

	em.close();
	emf.close();
    }
}
```

4. Pretpostavite da na poslužitelju postoji servlet koji odgovara na adresi: http://localhost:8080/app/racunaj?a=7&b=5 i koji uporabom formata JSON vraća sumu brojeva a i b, razliku brojeva a i b, kvadrat broja a i kvadrat broja b (dakle četiri podatka). Napišite HTML stranicu koja korisniku nudi mogućnost unosa brojeva a i b, uporabom tehnologije AJAX podatke Salje web-poslužitelju na izračun, te sva četiri primljena rezultata prikazuje Korisniku u HTML-dokumentu dinamičkom modifikacijom istoga. Vezano uz JavaScript, nije nužno da u slovo pogodite naziv svake od relevantnih metoda, već je bitno da način uporabe odgovara onome što nudi AJAX odnosno HTML.

```
<!DOCTYPE html>
<html>
	<head>
	</head>
	    <title>AJAX Kalkulator</title>
	<body>
	    <input type="number" id="a" name="a">
```

```
        <input type="number" id="b" name="b">
        <button onclick="izracunaj()">Izračunaj</button>

        <h2>Rezultati:</h2>
        <p id="suma"></p>
        <p id="razlika"></p>
        <p id="kvadratA"></p>
        <p id="kvadratB"></p>

        <script>
        function izracunaj() {
            var a = document.getElementById("a").value;
            var b = document.getElementById("b").value;

            fetch("http://localhost:8080/app/racunaj?a=" + a + "&b=" + b)
                .then(response => response.json())
                .then(data => {
                    document.getElementById("suma").textContent = "Suma: " + data.suma;
                    document.getElementById("razlika").textContent = "Razlika: " +
    data.razlika;
                    document.getElementById("kvadratA").textContent = "Kvadrat a: " +
    data.kvadratA;
                    document.getElementById("kvadratB").textContent = "Kvadrat b: " +
    data.kvadratB;
                });
        }
        </script>
    </body>
</html>
```

===== ZADACI =====

1. Rješavate u hw04 (pristup bazi kroz JDBC).

Proširite program tako da ga malo redefiniramo. Pretpostavite da se uz svaku stavku pamti broj ljudi koji su joj dali "like" (ono sto program pamti pod glasove smatrat ćemo ovom kategorijom), program treba pamtiti i broj ljudi koji su joj dali "dislike". Modificirajte web stranice na kojoj se glasa tako da stavke više nisu linkovi, već da uz svaku stavku korisnik može pritisnuti "like" ili "dislike" (korisnik može glasati, tj. stiskati "like" ili "dislike" više puta). Glavna stranica koja prikazuje rezultate u tablici treba prikazati broj glasova pod "like", broj glasova pod "dislike" te njihovu razliku (primjerice, ako neka stavke ima 5 "like" i 3 "dislike", uz stavku treba pisati 5, 3, 2). Ostatak programa (Excel, grafikon i slično) ne treba mijenjati.

2. Rješavate u hw05 (pristup bazi kroz JPA).

Proširite program tako da omogući da prijavljeni korisnik (dakle ne anoniman) kada je na stranici drugog autora njemu ostavi neku poruku, odnosno kada je na svojoj stranici i on može pisati poruke kada odgovara drugima. Pri tome radimo najtrivijalniji mogući podsustav poruka pa ne modeliramo niti pamtimo koja je poruka odgovor na koji već sve poruke prikazuje redoslijedom kojim su pisane, jednu ispod druge. Svi prijavljeni korisnici vide sve napisane poruke.

By: CrazyFreak (& ChatGPT)