PR8:

```
package com.example;

import javax.persistence.*;
```

Student.java Entity Class

```
@Entity
@Table(name = "student")  // Specifies the table name in the database
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)  // Auto-incremented ID
    @Column(name = "id")  // Maps to the 'id' column in the table
    private int id;

    @Column(name = "name")  // Maps to the 'name' column in the table
    private String name;

    @Column(name = "email")  // Maps to the 'email' column in the table
    private String email;

    // Constructors, getters, setters
    public Student() {
    }

    public Student(String name, String email) {
        this.name = name;
        this.email = email;
    }
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + ", email=" + email + "]";
    }
}
```

HibernateUtil.java

```
package com.example;


import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;


public class HibernateUtil {


    private static SessionFactory factory;


    static {

        // Create a SessionFactory using the configuration file

        factory = new 
Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactor
y();

    }


    public static Session getSession() {

        return factory.getCurrentSession();

    }


    public static void closeSessionFactory() {

        factory.close();

    }

}
```

Main class

```
package com.example;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

import java.util.List;

public class Main {

  public static void main(String[] args) {
    // Create a new Student instance
    Student newStudent = new Student("Alice", "alice@example.com");

    // Insert the student into the database
    try (Session session = HibernateUtil.getSession()) {
      // Start a transaction
      Transaction transaction = session.beginTransaction();

      // Save the student to the database
      session.save(newStudent);

      // Commit the transaction
      transaction.commit();

      System.out.println("Student saved successfully!");

      // Fetch all students from the database
      Query<Student> query = session.createQuery("from Student", Student.class);
```

```
            List<Student> students = query.getResultList();

            // Display the list of students
            for (Student student : students) {
                System.out.println(student);
            }
        }

        // Close the SessionFactory after use
        HibernateUtil.closeSessionFactory();
    }
}
```

Hibernate is a **Java-based ORM (Object-Relational Mapping)** framework that simplifies the interaction between Java applications and relational databases. ORM is a technique that allows developers to interact with a database using object-oriented programming (OOP) principles, mapping Java objects to database tables, eliminating the need for manual SQL query generation.

Hibernate automates many of the complexities of database interaction by providing:

- **Automatic table generation**.
- **Mapping Java objects to database tables**.
- **Support for transactions and caching**.

2. Key Concepts in Hibernate

- **SessionFactory**: The SessionFactory is the core object in Hibernate. It is responsible for creating Session objects, which are used to interact with the database. SessionFactory is created from the hibernate.cfg.xml configuration file, which contains the database connection settings and Hibernate-specific properties.
- **Session**: A Session object provides methods to interact with the database, such as saving, updating, deleting, and retrieving records. It acts as a bridge between the application and the database.
- **Transaction**: A Transaction in Hibernate is used to group a set of operations. It ensures atomicity by ensuring that either all operations are successfully committed or none are executed if an error occurs. Transactions are typically used in conjunction with Session.