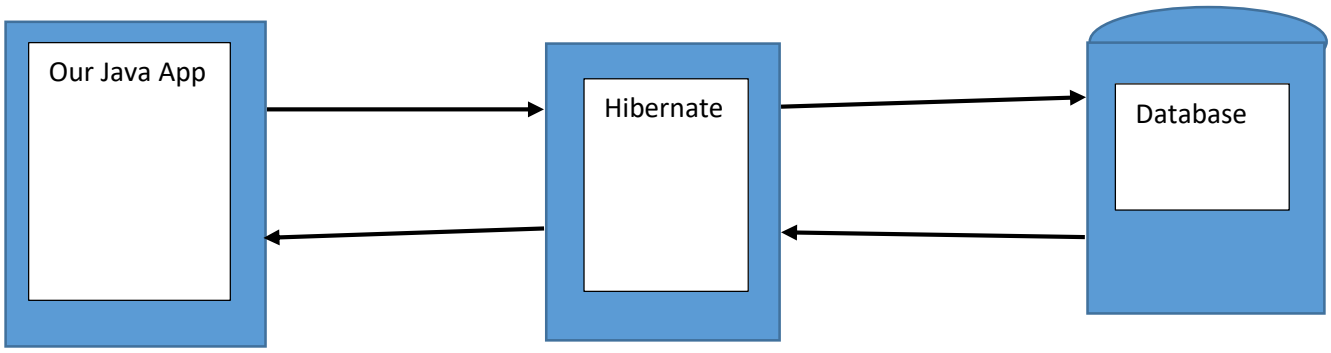
Hibernate/ JPA

Topics:

1. What is Hibernate?
2. Benefits of using hibernate
3. What is JPA?
4. Benefits of JPA
5. Code Snippets

What is Hibernate?

- A framework used for persisting/saving java objects in a database
- We can use it for saving & retrieving data from database.

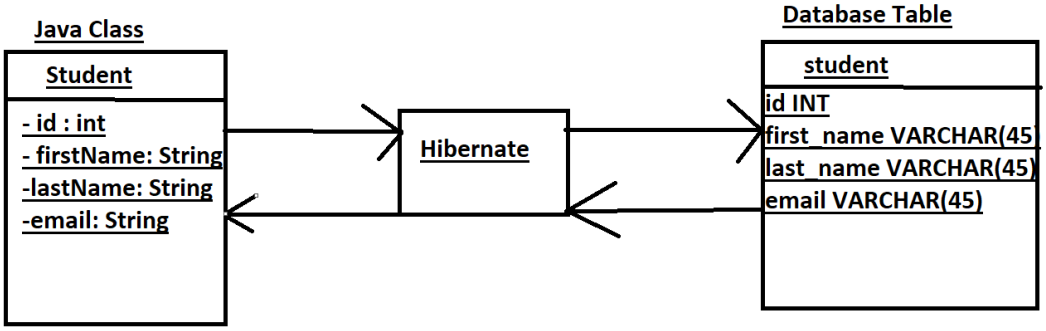


Benefits of Hibernate:

- It handles all of the low-level sql code
- Minimizes the amount of JDBC code we have develop
- Hibernate also provides the Object-to-Relational-Mapping(ORM)

ORM:

- As a developer all we need to do is tell hibernate how our java class or object maps to the database.



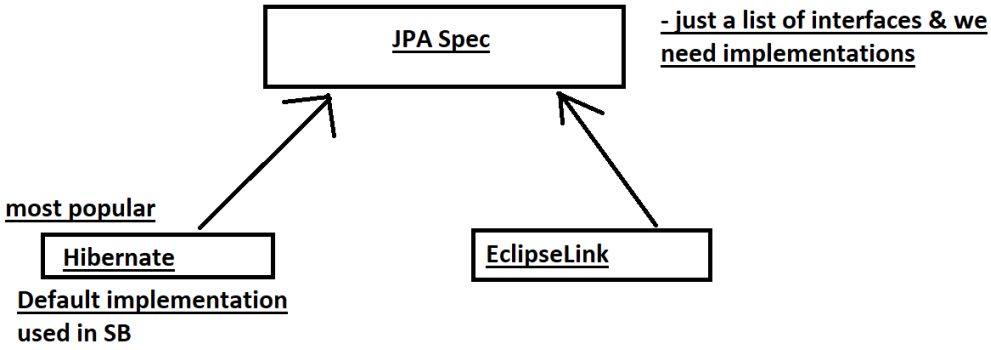
- What we will do is Map this Java class to the given table &
- We set up one-to-one mapping between the fields and actual columns in the database
- You can set up this mapping via configuration file using XML, but we are going to use Java Annotations

What is JPA?

- Jakarta-Persistence-API …… previously known as Java Persistence API
- Standard API for Object-to-relational Mapping

- Only a Specification
- Defines a set of interfaces
- Requires an implementation to be usable

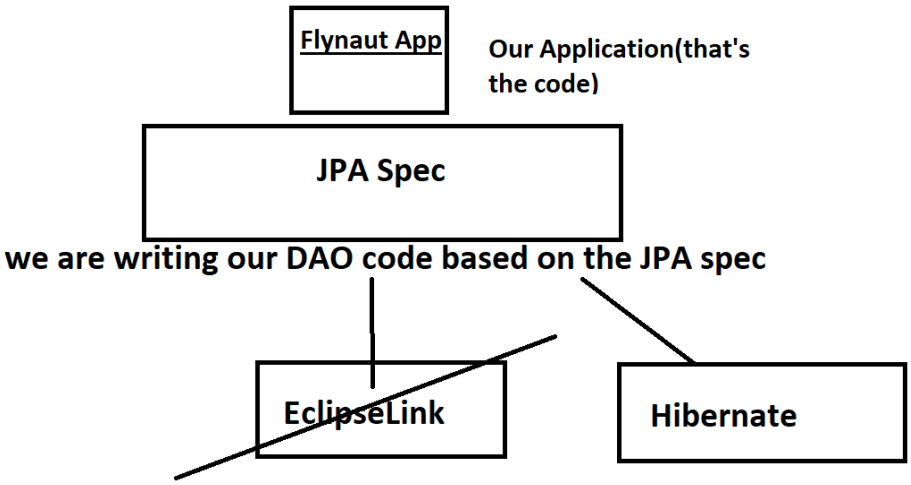
- JPA-vendor implementations



What are benefits of using JPA?

- By having standard API, we are not locked to vendors implementations.
- Can theoretically switch vendor implementations
  -Ex. If Vendor ABC stops supporting their product
  We can switch to another vendor

- Example of swapping vendor implementations



- Simply by changing the configuration we can change the vendor.

Terminologies of JPA*

Quick Example:

Saving a java object with JPA

// Create a java object

Student theStudent = new Student(“Krishna”, “Jain”, “k@gmail.com”);

//save it to db

entityManager.persist(theStudent);

Special JPA helper object

- BTS hibernate is the implementation of JPA
  But here JPA with the hibernate does all the work for us in background.

- Retrieving a java object with JPA
  //create java object
  //save it to db
  //now retrieve from db using primary key
  int theId=1;
  Student myStudent=entityManager.find(Student.class,theId);

- JPA/Hibernate CRUD Apps
- Create objects
- Read objects
- Update objects
- Delete objects

- Relation of Hibernate/JPA and JDBC

  How does JPA relate to JDBC?

- Hibernate/JPA uses JDBC for all the db communications.

Setting Up SB project:

1. Start.spring.io
2. Add dependencies(MySQL driver, Spring Data JPA)
3. Generate

```
spring-boot-starter-data-jpa -Spring Data JPA
  mysql-connector-j  - MySQL Driver
```

Auto Configuration:

- SB will read DB connection information from application.properties file

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/student_tracker
spring.datasource.username=springstudent
spring.datasource.password=springstudent
```

No need to give jdbc driver class name

SB will automatically detect it based on URL

- JPA Dev Process
  1. Annotate class
  2. Develop Java code to perform db operations

Entity Class:

Java Class that is mapped to a db table.

IMPs:

1. It must be annotated with @Entity

2. It must have a public or protected no-arg constructor

Java Annotations

Step 1. Map class to database table

Step 2. Map fields to database columns

1.

```
@Entity
@Table(name=”student”)
public class Student{……}
```

2.

```
@Entity
@Table(name=”student”)
public class Student{
      @Id
      @GeneratedValue(strategy=GenerationType.IDENTITY)
      @Column(name=”id”)
      private int id;
     @Column(name=”first_name”)
     private String firstName;

}
```

GenerationTypes:

AUTO

IDENTITY-Assign primary keys using db identity column

SEQUENCE

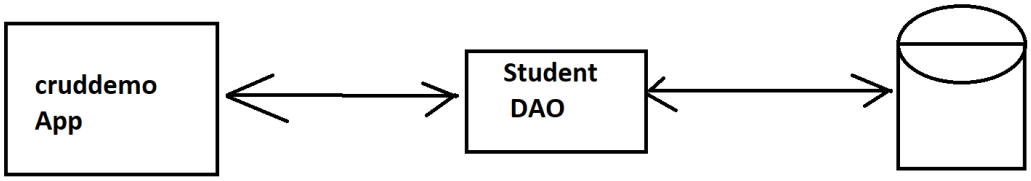
TABLE

- Saving a java object

  Sample app features

- Create a new Student
- Read a Student
- Update a Student
- Delete a Student

- Student Data Access Object
- Responsible for interacting with the db
- This is a common design pattern : Data Access Object(DAO)



Our Student data access object will have a number of methods

1. save(…) -> for saving a student
2. findById(…)
3. findAll(…)
4. findByLastName(…)
5. update(…)
6. delete(…)
7. deleteAll()

Dev Process:

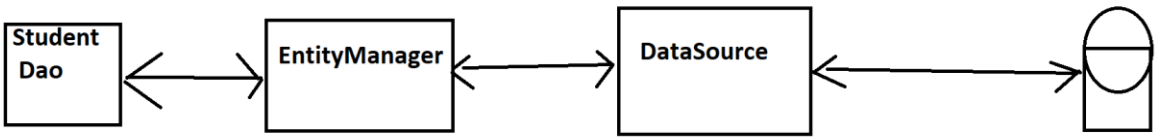
Student DAO

1. Define DAO interface

2. Define DAO implementation

    - Inject Entity Manager

3. update main app



Spring Data JPA - Reference Documentation

1.

```
Public interface StudentDAO{
        void save(Student theStudent);
}
```

2.

```
Public class studentDAOImpl implements StudentDAO{
private EntityManager entityManager;
@Autowired
public StudentDAOImpl(EntityManager theEntityManager){
                entityManager=theEntityManager;
}

@Override
Public void save(Student theStudent){
entityManager.persist(theStudent);   // save the object
        }
}
```

Spring @Transactional

- Automatically begin & end a transaction for our JPA code
- No need to explicitly do this in our code