

# DATABASE ACCESS WITH HIBERNATE/JPA

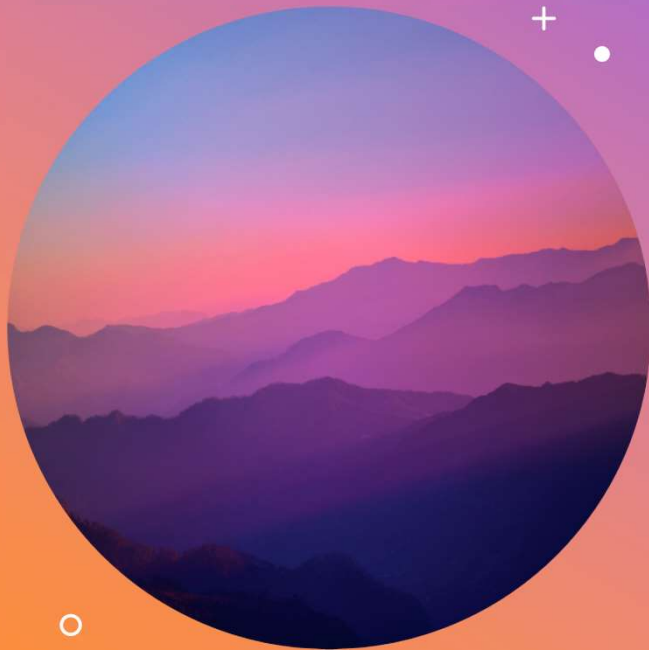


Nuttachai Kulthammanit



# HIBERNATE / JPA OVERVIEW

Chapter 1



# TOPICS

Hibernate คืออะไร?

ประโยชน์ของ Hibernate

JPA คืออะไร?

ประโยชน์ของ JPA

Code Snippets

# Hibernate คืออะไร?

- Framework สำหรับการบันทึก Java Object ใส่น Database
- ทำได้ทั้ง อ่าน เขียน เปลี่ยนแปลงและลบข้อมูลผ่าน Java Object แล้ว Hibernate จะไปจัดการให้จริง ๆ ใน Database
  - <https://hibernate.org/orm/>

# Hibernate คืออะไร?

- Framework สำหรับการบันทึก Java Object ไว้ใน Database
- ทำได้ทั้ง อ่าน เขียน เปลี่ยนแปลงและลบข้อมูลผ่าน Java Object แล้ว Hibernate จะไปจัดการให้จริง ๆ ใน Database
  - <https://hibernate.org/orm/>



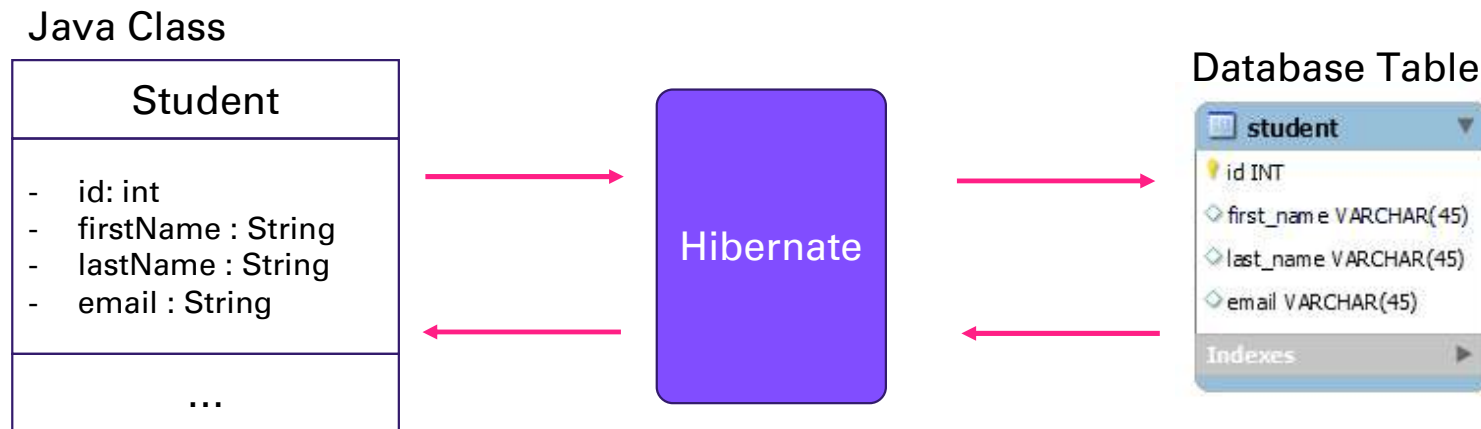
# ประโยชน์ของ Hibernate

- **Hibernate** จะจัดการในส่วนของ **low-level** ที่เป็น **SQL** ให้เราทั้งหมด
- ทำให้เราไม่ต้องเขียนโค้ดเยอะเวลาพัฒนา **Application**
- **Hibernate** ทำให้เราสามารถใช้ **Object-to-Relational Mapping (ORM)**
  - ทำให้เราสามารถจัดการข้อมูลใน **Database** ได้ง่าย
  - เราสามารถทำผ่าน **Java Object** ได้เลย ไม่ว่าจะเป็นการ อ่าน เปลี่ยนแปลง ลบหรือสร้างข้อมูล
  - สุดท้ายแล้ว **Hibernate** จะไปทำให้เราจริง ๆ อีกทีใน **Database**



# Object-To-Relational Mapping (ORM)

- การที่เราจะทำแบบนั้นได้เราต้องกำหนด **Mapping** ระหว่าง **Java class** กับ **database Table** ก่อน



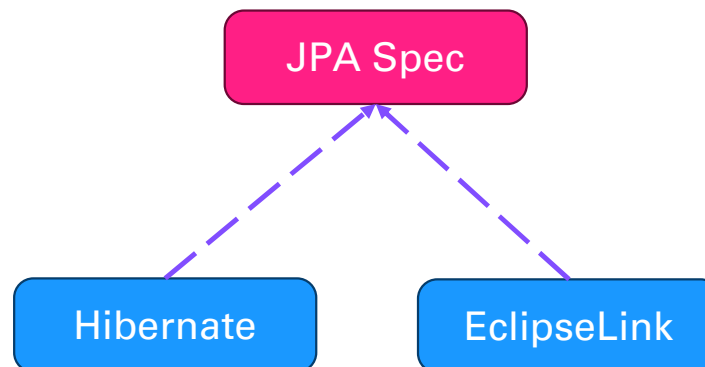
# JPA คืออะไร

- Jakarta Persistence API (JPA) – เมื่อก่อนชื่อ Java Persistence API
  - เป็น Standard API สำหรับการทำ Object-to-Relational-Mapping (ORM)
- มีแค่ Specification
  - เป็นแค่ชุดของ Interface
  - แต่ต้องมี Implementation เพื่อให้มันทำงานได้
- อ่านเพิ่มเติม: <https://www.jcp.org/en/jsr/detail?id=338>



# JPA – Vendor Implementation

- JPA เป็น interface ที่ให้ Implement เพื่อให้สามารถใช้งานได้
- ก็มีหลายยี่ห้อเลยที่ทำ Implementation ขึ้นมาให้ใช้
- ซึ่ง Hibernate และ EclipseLink ก็คือหนึ่งในนั้น (แต่ Hibernate Popular สุด)
- แล้วก็เป็น default implementation ที่ใช้ใน spring boot
- [https://en.wikipedia.org/wiki/Jakarta\\_Persistence](https://en.wikipedia.org/wiki/Jakarta_Persistence)

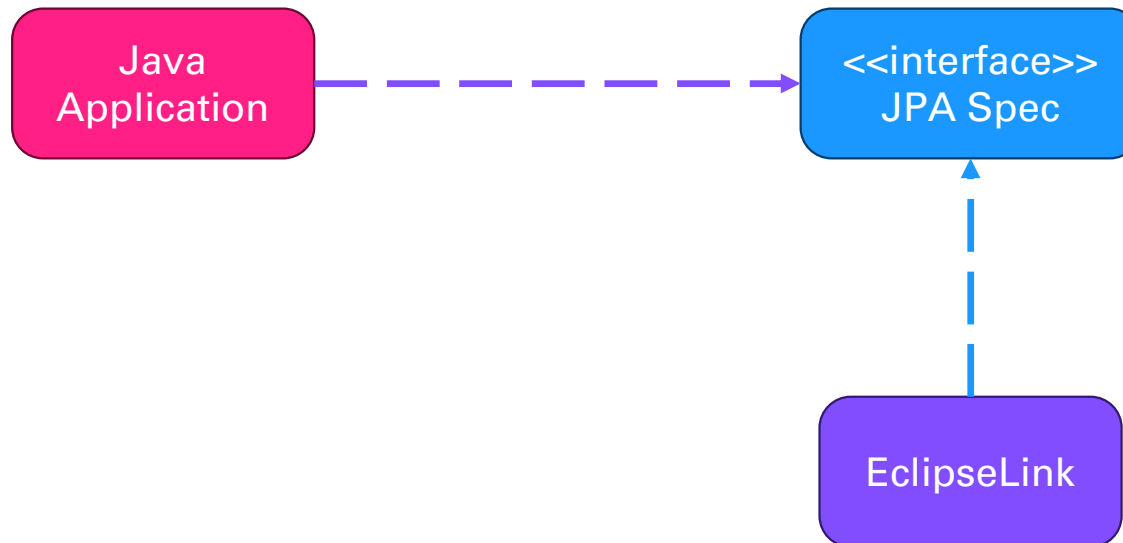


# ประโยชน์ของ JPA

- การที่มี **Standard API** ทำให้เราไม่ต้องยึดกับ **Vendor** ใด **vendor** หนึ่ง
- ทำให้มันมีความ **portable, flexible code** โดย **JPA spec (interface)**
- ซึ่งสามารถทำให้สลับ **vendor implementation** ได้
  - สมมุติวันหนึ่ง **Vendor ABC** เลิก **support** ขึ้นมา
  - เราก็สามารถสลับไปใช้ของ **Vendor XYZ** ได้

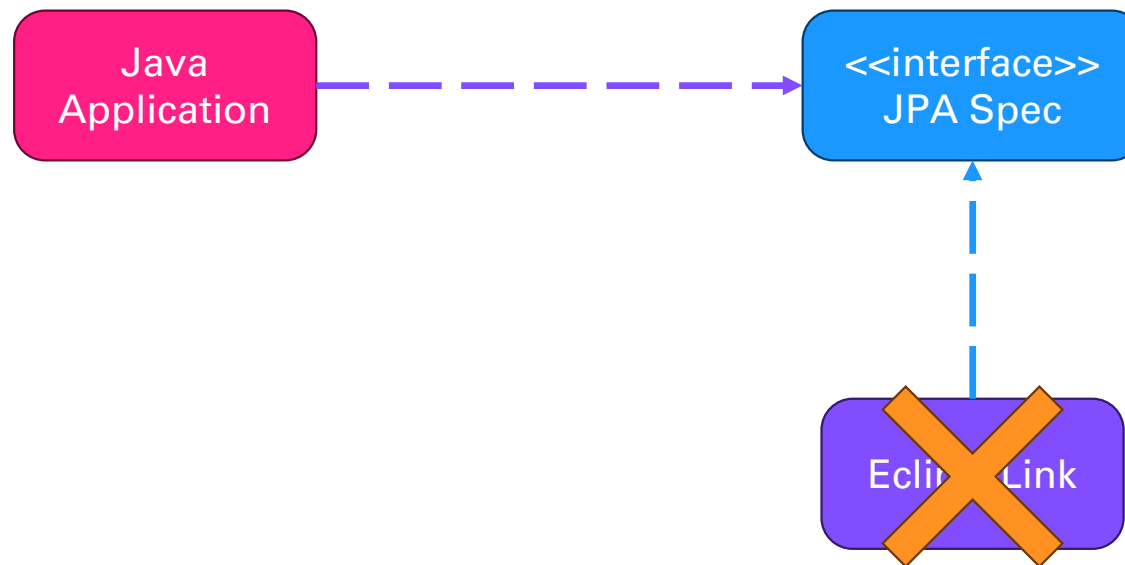
# ประโยชน์ของ JPA - ตัวอย่าง

- สมมติตอนนี้เรากำลังใช้ EclipseLink อยู่
- ซึ่ง EclipseLink ก็ Implement ตัว JPA Spec อยู่



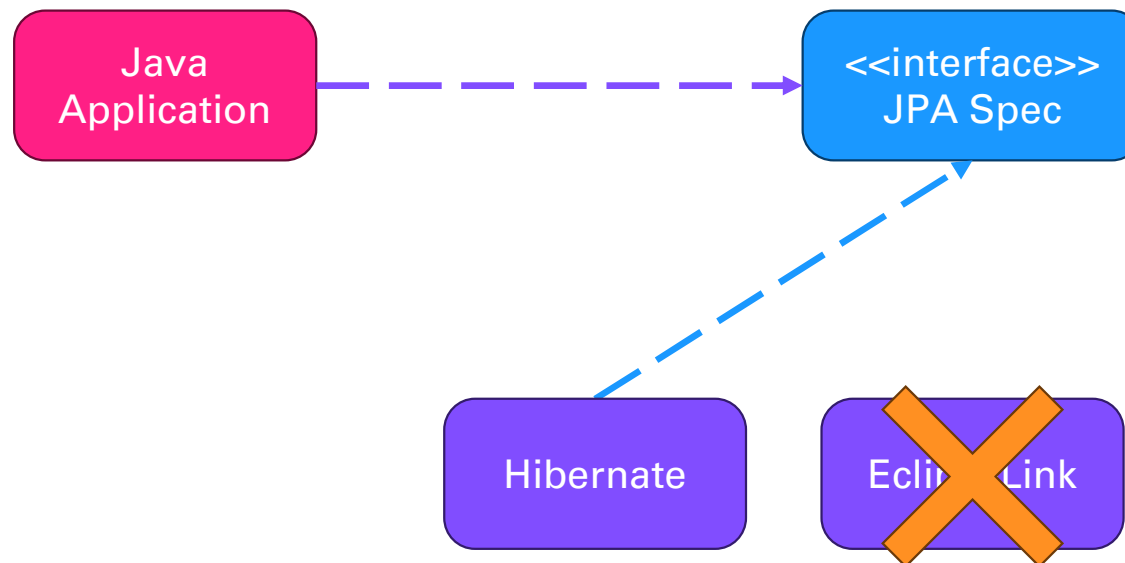
# ประโยชน์ของ JPA - ตัวอย่าง

- แล้ววันนี้ EclipseLink เลิก Support แล้ว



# ประโยชน์ของ JPA - ตัวอย่าง

- เราก็แค่เปลี่ยน Config มาใช้ตัว Hibernate แทน
- เพราะว่าทั้ง Hibernate และ EclipseLink มีการ implement ผ่าน JPA Spec ทั้งคู่ใน Java Application เลยไม่ต้องมีการเปลี่ยนโค้ดเลย



# การบันทึก Java Object ด้วย JPA - ตัวอย่าง

- **entityManager** เป็น JPA Helper Object

```
// Create Java object
Student student = new Student("Somchai", "Meeboon", "somchai.me@hotmail.com");

// Save it to database
entityManager.persist(student);
```

ตัวข้อมูลก็จะถูก Save ลง Database  
โดยใช้ SQL insert (อันนี้มีคนทำให้)  
ถ้าเราใช้ Hibernate คนที่ทำก็คือ Hibernate

# การบันทึก Java Object ด้วย JPA - ตัวอย่าง

- **entityManager** เป็น JPA Helper Object

ในสมัยก่อนที่ยังไม่มี JPA กับ Hibernate  
เราต้องเขียน SQL Code เอง

```
// Create Java object
Student student = new Student("Somchai", "Meeboon", "somchai.me@hotmail.com");

// Save it to database
entityManager.persist(student);
```

ตัวข้อมูลก็จะถูก Save ลง Database  
โดยใช้ SQL insert (อันนี้มีคนทำให้)  
ถ้าเราใช้ Hibernate คนที่ทำก็คือ Hibernate

# การอ่าน Java Object ด้วย JPA - ตัวอย่าง

- เราจะดึง Student ที่เพิ่ง Save ไปออกมา โดยใช้ primary key (สมมติให้เป็น 1)
  - เบื้องหลังตัว Implementation (Hibernate) จะไปใช้ SQL Select Query ออกมา
  - แล้วเอามาใส่ใน Student Object ให้เรา

```
// Create Java object
Student student = new Student("Somchai", "Meeboon", "somchai.me@hotmail.com");

// Save it to database
entityManager.persist(student);

// retrieve data from database using primary key
int id = 1;
Student myStudent = entityManager.find(Student.class, id);
```



# การ query Java Object ด้วย JPA - ตัวอย่าง

- อันนี้คือหาทั้งหมดจาก Student Table
  - แล้วตัว Implementation (Hibernate) จะแปลงเป็น List<Student> ให้เรา

```
TypeQuery<Student> query = entityManager.createQuery("from Student", Student.class);  
List<Student> students = query.getResultList();
```

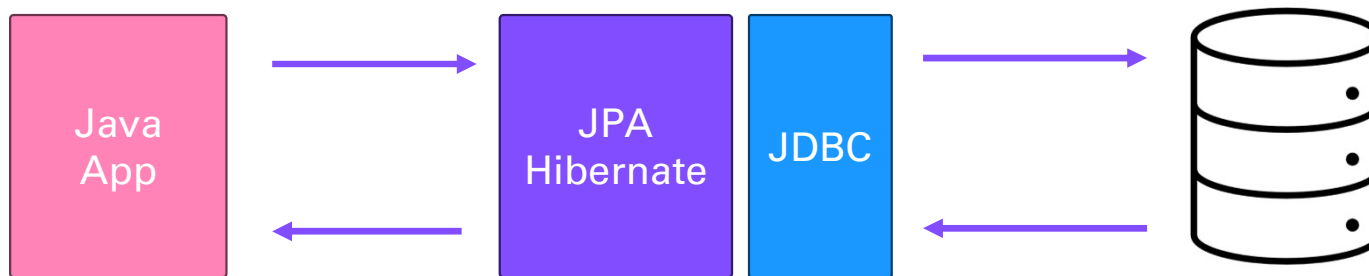
# JPA/Hibernate CRUD

- Create object
- Read object
- Uppdate object
- Ddelete object



# JPA/Hibernate กับ JDBC

- Hibernate/JPA ใช้ JDBC ในการติดต่อกับ Database ทั้งหมด
- Hibernate/JPA ก็เป็นอีก Layer หนึ่งที่เขียน on top บน JDBC
  - พูดย่าง ๆ คือเบื้องหลังของ Hibernate/JPA ก็ใช้ JDBC



# MAPPING CLASS TO DATABASE TABLE

Chapter 2

# Setup Database

Mini-workshop: Setup Database

Link: <https://raw.githubusercontent.com/soncomqiq/kbtg-spring-and-react/main/config%20props.txt>

Link: <https://raw.githubusercontent.com/soncomqiq/kbtg-spring-and-react/main/create-student-table.sql>

Ref: lab-s4-l1

# Setting Up Project

- ใน Spring Boot, Hibernate เป็น default implementation ของ JPA
- entityManager เป็นตัวหลักที่ใช้ในการสร้าง queries
- entityManager มาจาก JPA
- Based on config, Spring Boot จะสร้าง Bean พวกนี้ให้เราอัตโนมัติ
  - Datasource, EntityManager, ...
- หลังจากนั้นเราก็สามารถ Inject Bean พวกนี้ไปที่ Application เราได้

# Setting Up Project

- การใช้ Spring Initializr
- ไปที่ Spring Initializr website, [start.spring.io](https://start.spring.io)
- เพิ่ม dependencies
  - MySQL Driver: `mysql-connector-j`
  - Spring Data JPA: `spring-boot-starter-data-jpa`

# Setting Up Project

- หลังจากเราโหลดมา Import แล้ว Spring Boot จะ auto config ตัว data source ให้เรา
- โดยอ้างอิงจากไฟล์ pom
  - JDBC Driver: mysql-connector-j
  - Spring Data (ORM): spring-boot-starter-data-jpa
- Config สำหรับการเชื่อมต่อ Database จะอ่านจาก application.properties



# Setting Up Project

- application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/database_name  
spring.datasource.username=your_db_username  
spring.datasource.password=your_db_pwd
```

- ไม่ต้องใส่ JDBC driver class name เดี่ยว Spring Boot มัน detect ได้เองจาก URL

# Command Line App

- เดี่ยวเราจะสร้าง Spring Boot – Command Line App ขึ้นมา
- ตัวนี้จะทำให้เรา Focus แค่ตัว code ที่เป็น Hibernate/JPA
- แล้วเดี๋ยวหลังจากเราทำ Hibernate/JPA เป็นแล้ว
- เราค่อยย้ายไปทำในตัวที่เป็น CRUD REST API (พวก Controllers)

# Command Line App

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(String[] args) {
        return runner -> {
            System.out.println("Hello World");
        };
    }
}
```

Code นี้จะรันหลังจาก Spring Bean ทั้งหมด Load เสร็จแล้ว

# Command Line App

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(String[] args) {
        return runner -> {
            System.out.println("Hello World");
        };
    }
}
```

เราสามารถใส่ Code ของเราให้  
มันทำอะไรก็ได้ใน Lambda  
expression นี้

Code นี้จะรันหลังจาก Spring  
Bean ทั้งหมด Load เสร็จแล้ว

A vertical bar on the left side of the slide with a gradient from orange at the top to blue at the bottom.

# Setting Up Project

Mini-workshop: Setting Up Project

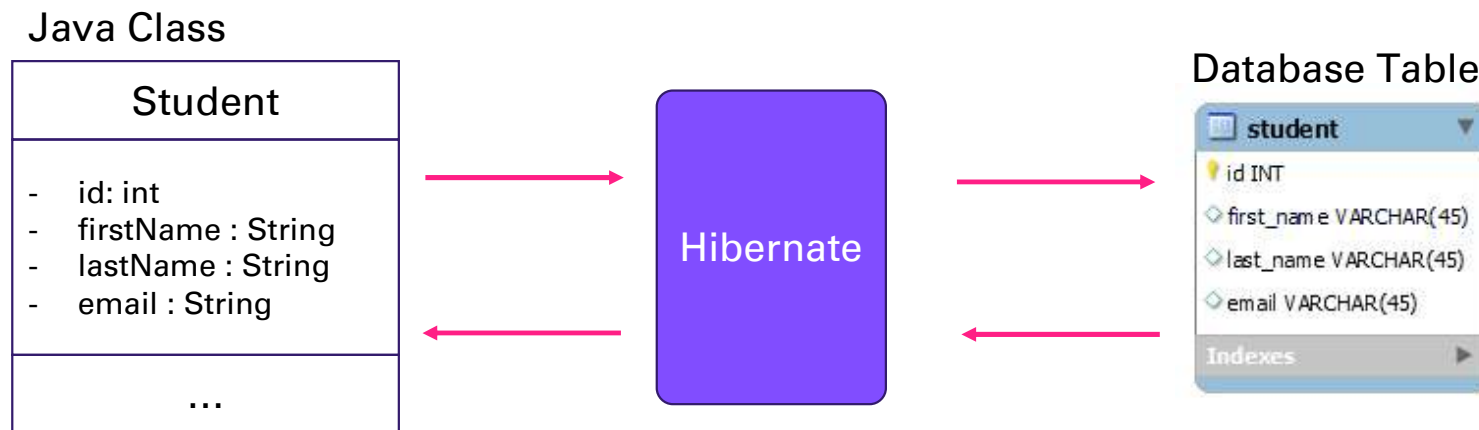
Ref: lab-s4-l2

# JPA Annotation

- วิธีการเชื่อมต่อกับ Database
  1. ใส่ Annotation ที่ Java Class
  2. ใช้ Code ในการจัดการ Database

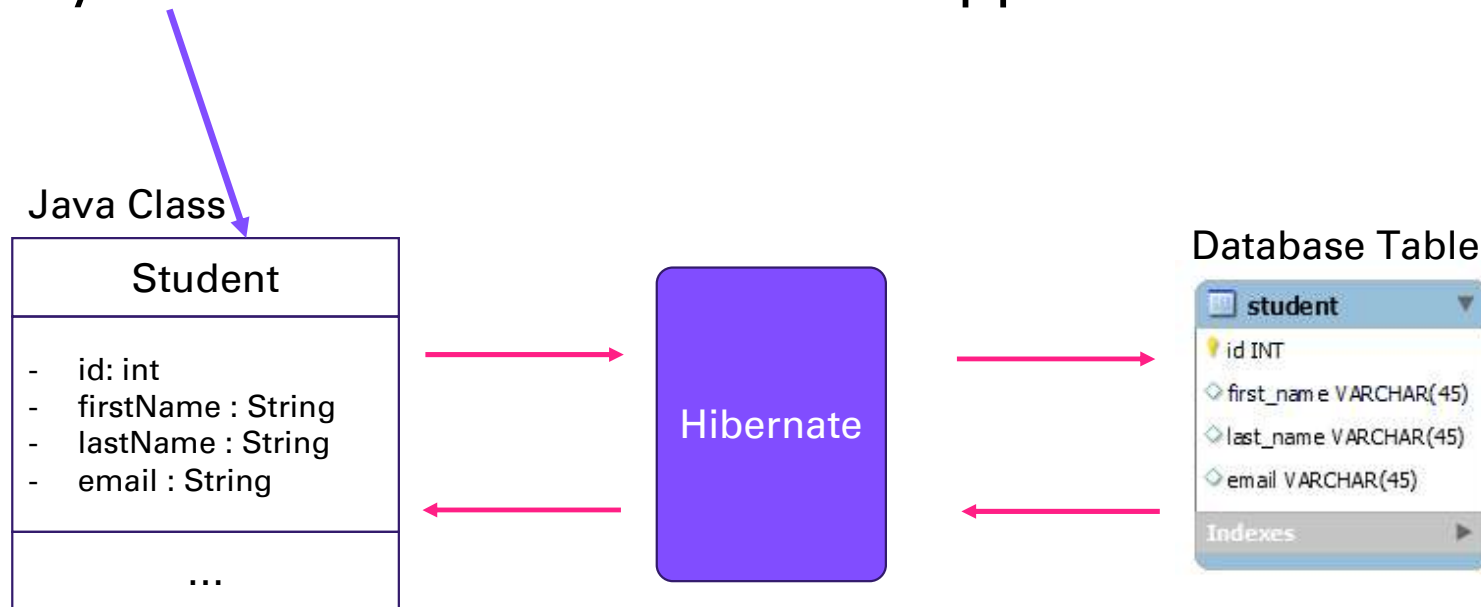
## คำศัพท์เพิ่มเติม

- Entity Class คือ Java Class ที่ใช้ในการ Mapped ไปที่ตารางใน Database



## คำศัพท์เพิ่มเติม

- **Entity Class** คือ Java Class ที่ใช้ในการ Mapped ไปที่ตารางใน Database





# Entity Class

- อย่างน้อยใน Entity Class
  - ต้องมี @Entity Annotation
  - ต้องมี public หรือ protected constructor ที่ไม่มี argument
    - สามารถมีแบบที่มี Argument ได้
    - แต่ต้องมีแบบ no-argument ด้วย

# Java Annotation

การ Setup ตัว Entity ให้ Map กับตารางใน Database

1. Map ตัว Class กับตารางใน Database
2. Map ตัว Fields ให้กับคอลัมน์ใน Database

# Map ตัว Class กับตารางใน Database

ต้องใส่ชื่อตารางใน Database ใน Annotation @Table

Java Class

```
@Entity
@Table(name = "student")
public class Student {
    ...
}
```

Java Class

Student
- id: int - firstName : String - lastName : String - email : String
...

Database Table

student
id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(45)
Indexes

# Map ตัว Fields ให้กับคอลัมน์ใน Database

ต้องใส่ชื่อคอลัมน์ในตารางของ **student** ใน **@Column**

## Java Class

```
@Entity
@Table(name = "student")
public class Student {

    @Id
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;

    ...
}
```

## Java Class

### Student

- id: int
- firstName : String
- lastName : String
- email : String

...

## Database Table

student
id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(45)
Indexes

# Map ตัว Fields ให้กับคอลัมน์ใน Database

ต้องใส่ชื่อคอลัมน์ในตารางของ **student** ใน **@Column**

## Java Class

```
@Entity
@Table(name = "student")
public class Student {

    @Id
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;

    ...
}
```

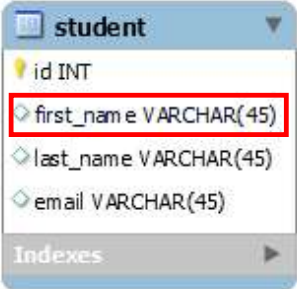
## Java Class

### Student

- id: int
- firstName : String
- lastName : String
- email : String

...

## Database Table



student
id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(45)
Indexes

# MySQL – Auto Increment

Id เป็น primary key และเพิ่มขึ้นอัตโนมัติ

```
CREATE TABLE student (  
    id int NOT NULL AUTO_INCREMENT,  
    first_name varchar(45) DEFAULT NULL,  
    last_name varchar(45) DEFAULT NULL,  
    email varchar(45) DEFAULT NULL,  
    PRIMARY KEY (id)  
)
```

# JPA Identify – Primary Key

```
@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;

}
```

เป็นบอกว่าเป็น ID หรือ Primary

# JPA Identify – Primary Key

```
@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;

}
```

GenerationType เป็น Identify  
คือ ID Value นี้จะถูก  
Generated และ จัดการโดย  
Database



# Mapping Class to database table

Mini-workshop: Mapping Class to database table

Ref: lab-s4-l3

# DATABASE CRUD WITH JPA/HIBERNATE

Chapter 3

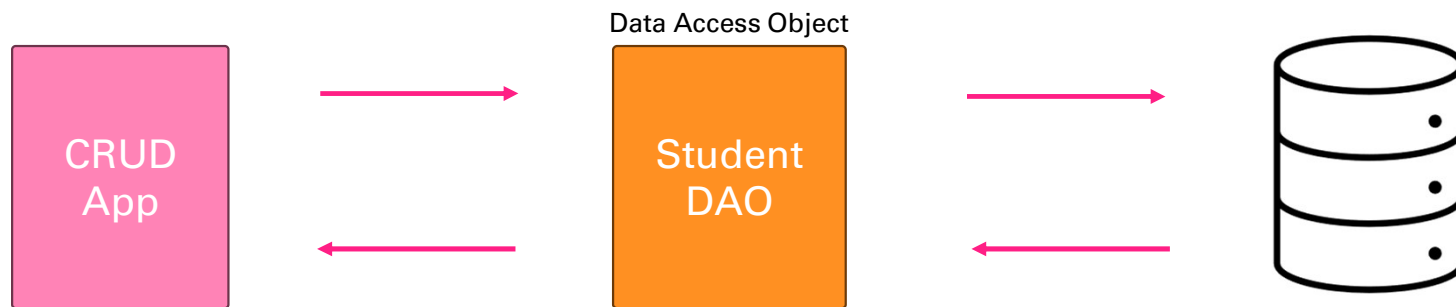
# ตัวอย่างของ App

เดี๋ยวเราจะทำ App ที่มี 4 ฟังก์ชันนี้

- **Create a new Student**
- **Read a Student**
- **Uppdate a Student**
- **Ddelete a Student**

# Student Data Access Object

- เป็นตัวที่ทำหน้าที่ติดต่อกับ Database
- เป็น Design Pattern แบบหนึ่ง: Data Access Object (DAO)

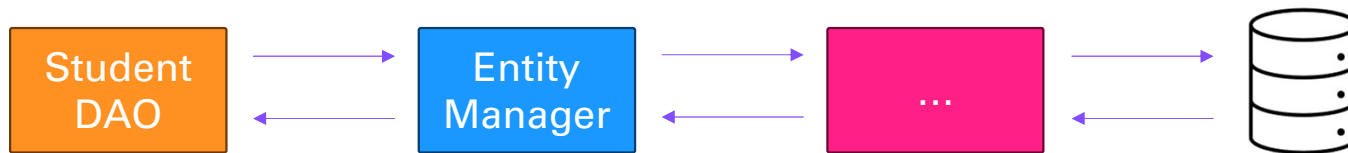


# Student Data Access Object

- Method ใน DAO
  - save(...)
  - findById(...)
  - findAll()
  - findByLastName(...)
  - update(...)
  - delete(...)
  - deleteAll()

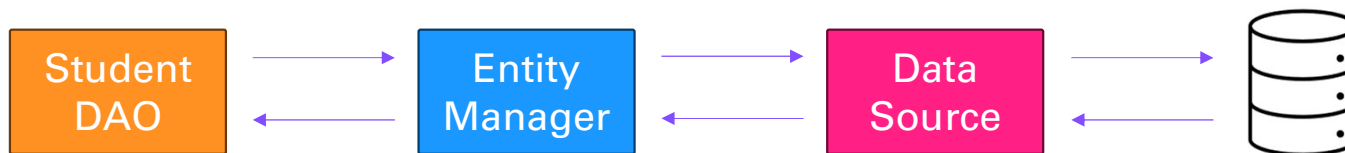
# Student Data Access Object

- DAO ต้องการ JPA Entity Manager
- JPA Entity Manager เป็นตัวหลักที่ใช้ในการบันทึกหรือดึง Entity ออกมาใช้



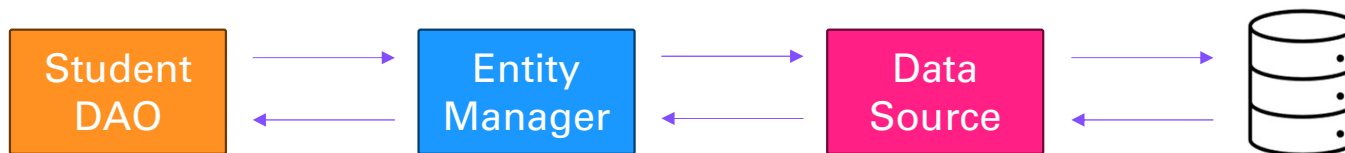
# JPA Entity Manager

- JPA Entity Manager ต้องการ Data Source
- ตัว Data Source เป็นคนสร้างการเชื่อมต่อกับ Database
- ทั้ง JPA Entity Manager และ Data Source ถูกสร้างโดย Spring Boot
  - โดยจะอ้างอิงจากไฟล์: application.properties (JDBC URL, username, password เป็นต้น)
- แล้วเราก็สามารถ autowire/inject ตัว JPA Entity Manager เข้าไปใน Student DAO ได้เลย



# Student DAO

- วิธีการทำ
  1. สร้าง DAO interface
  2. สร้าง DAO implementation
    - Inject ตัว Entity Manager
  3. เรียกใช้ใน Class ที่เราต้องการ





# 1. สร้าง DAO interface

เราสร้าง interface ที่มี save method ที่รับ student (entity class)

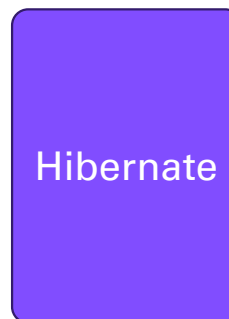
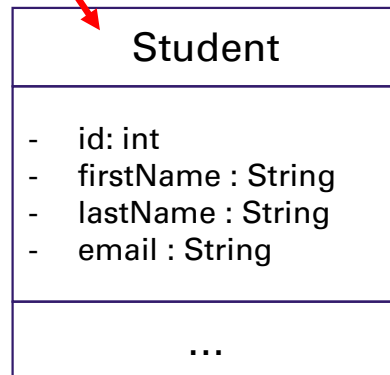
```
public interface StudentDAO {  
→ void save(Student student);  
}
```

# 1. สร้าง DAO interface

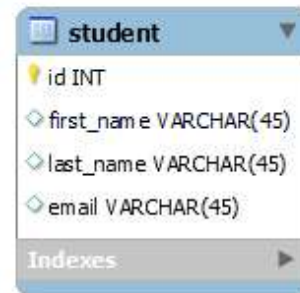
เราสร้าง interface ที่มี save method ที่รับ student (entity class)

```
public interface StudentDAO {  
    void save(Student student);  
}
```

Java Class



Database Table



## 2. สร้าง DAO implementation

แล้วเราก็สร้าง Implementation ที่ implement ตัว interface จากขั้นตอนแรก

```
public class StudentDAOImpl implements StudentDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public StudentDAOImpl(EntityManager entityManager){  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    public void save(Student student) {  
  
    }  
}
```

## 2. สร้าง DAO implementation

แล้วเราก็สร้าง Implementation ที่ implement ตัว interface จากขั้นตอนแรก

```
public class StudentDAOImpl implements StudentDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public StudentDAOImpl(EntityManager entityManager){  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    public void save(Student student) {  
        entityManager.persist(student);  
    }  
}
```

เรามี field ที่เป็น EntityManager ที่ใช้ในการ  
จัดการ Entity

แล้วเราก็ inject ตัว EntityManager มาให้  
Implementation นี้ใช้

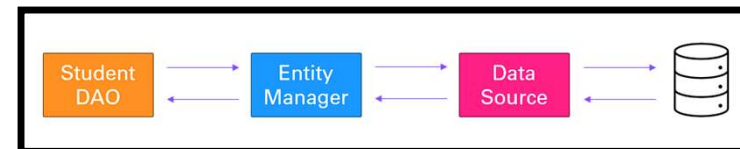
## 2. สร้าง DAO implementation

แล้วเราก็สร้าง Implementation ที่ implement ตัว interface จากขั้นตอนแรก

```
public class StudentDAOImpl implements StudentDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public StudentDAOImpl(EntityManager entityManager){  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    public void save(Student student) {  
        entityManager.persist(student);  
    }  
}
```

เรามี field ที่เป็น EntityManager ที่ใช้ในการ  
จัดการ Entity

แล้วเราก็ inject ตัว EntityManager มาให้  
Implementation นี้ใช้



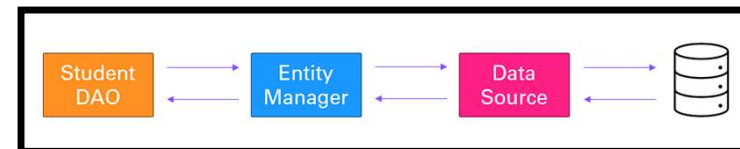
## 2. สร้าง DAO implementation

แล้วเราก็สร้าง Implementation ที่ implement ตัว interface จากขั้นตอนแรก

```
public class StudentDAOImpl implements StudentDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public StudentDAOImpl(EntityManager entityManager){  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    public void save(Student student) {  
        entityManager.persist(student);  
    }  
}
```

เรามี field ที่เป็น EntityManager ที่ใช้ในการ  
จัดการ Entity

แล้วเราก็ inject ตัว EntityManager มาให้  
Implementation นี้ใช้



Save ตัว Java Object

ซึ่งเราก็ต้องส่ง Student (ที่เป็น Entity Class) เข้าไป

แล้วเดี๋ยว JPA/Hibernate ก็จะไปเอา Student Object ตัวนี้ไป Save  
ลง Database เอง

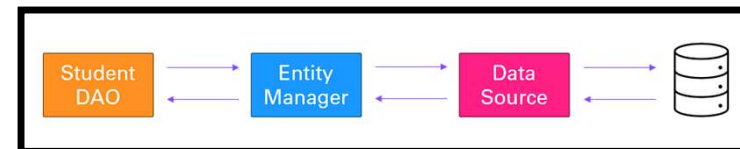
## 2. สร้าง DAO implementation

แล้วเราก็สร้าง Implementation ที่ implement ตัว interface จากขั้นตอนแรก

```
public class StudentDAOImpl implements StudentDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public StudentDAOImpl(EntityManager entityManager){  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    public void save(Student student) {  
        entityManager.persist(student);  
    }  
}
```

เรามี field ที่เป็น EntityManager ที่ใช้ในการ  
จัดการ Entity

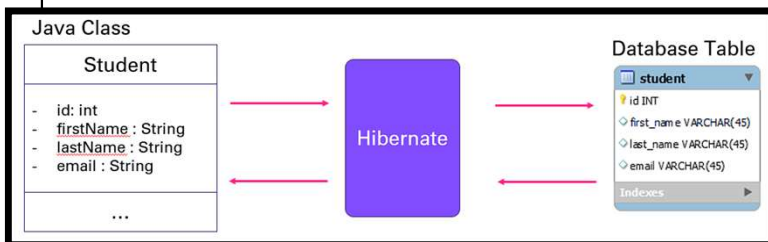
แล้วเราก็ inject ตัว EntityManager มาให้  
Implementation นี้ใช้



Save ตัว Java Object

ซึ่งเราก็ต้องส่ง Student (ที่เป็น Entity Class) เข้าไป

แล้วเดี๋ยว JPA/Hibernate ก็จะไป Save  
ลง Database เอง



## 2. สร้าง DAO implementation

```
public class StudentDAOImpl implements StudentDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public StudentDAOImpl(EntityManager entityManager) {  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    @Transactional  
    public void save(Student student) {  
        entityManager.persist(student);  
    }  
}
```

Spring จะช่วยจัดการเรื่อง  
Transaction ให้เรา

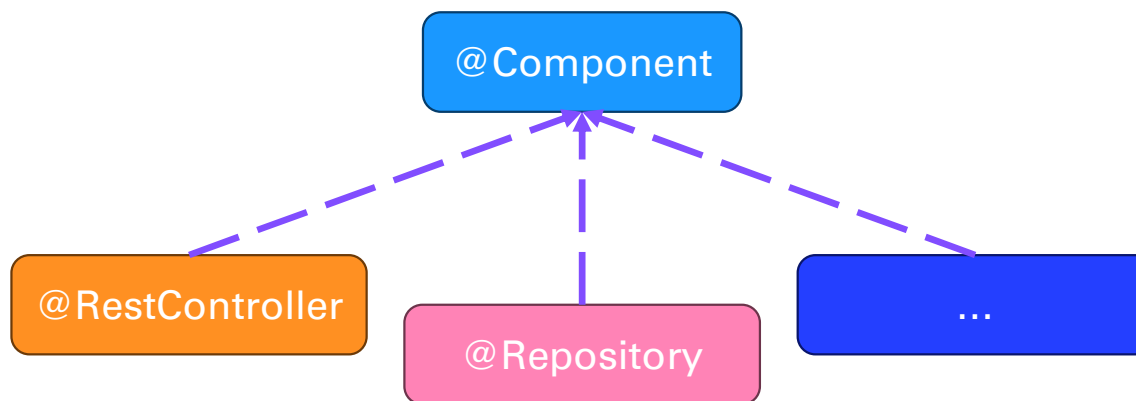


# Spring @Transactional

- Spring มี @Transactional Annotation ให้ใช้
- มันจะทำการเปิดและปิด Transaction ให้เราโดยอัตโนมัติ (JPA code)
  - ปกติเราต้องมานั่งเรียกฟังก์ชันสำหรับการเปิด Transaction และการปิด Transaction เอง ใน Code
  - แต่ถ้าเราใช้ อันนี้ เราไม่ต้องเขียนเลย
- Spring จะจัดการให้เองอัตโนมัติเลย

# Annotation พิเศษสำหรับ DAO

- Spring มี @Repository annotation ให้เราใช้
- ซึ่งก็เป็น Annotation ที่ Extends มาจาก @Component
- สำหรับใช้กับ DAO
- Spring จะ Register ตัว DAO Implementation ให้เรา
- รวมถึงจัดการแปลง (Translates) ตัว JDBC Exception ให้เราใช้ได้ง่ายขึ้นด้วย



## 2. สร้าง DAO implementation

Register ตัว DAO  
implementation ให้เรา

```
@Repository
public class StudentDAOImpl implements StudentDAO {

    private EntityManager entityManager;

    @Autowired
    public StudentDAOImpl(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    @Override
    @Transactional
    public void save(Student student) {
        entityManager.persist(student);
    }
}
```

### 3. เรียกใช้ใน Class ที่เราต้องการ

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
        return runner -> {
            createStudent(studentDAO);
        };
    }

    private void createStudent(StudentDAO studentDAO) {
        // create a student object
        System.out.println("Create new student object...");
        Student student = new Student("Hello", "World", "hello.w@hotmail.com");

        // save the student object
        System.out.println("Saving the student ...");
        studentDAO.save(student);

        // display id of the saved student
        System.out.println("ID: " + student.getId());
    }
}
```

Inject ตัว StudentDAO



# บันทึก Java Object ด้วย JPA

Mini-workshop: บันทึก Java Object ด้วย JPA

Ref: lab-s4-l4

# JPA/Hibernate CRUD

- Create object
- Read object
- Uppdate object
- Ddelete object



# การเรียกข้อมูล Java Object

```
// retrieve/read from database using a primary key  
// in this example, retrieve Student with primary key: 1
```

```
Student mystudent = entityManager.find(Student.class, 1);
```



Entity Class

Primary Key

# การเรียกข้อมูล Java Object

*// retrieve/read from database using a primary key  
// in this example, retrieve Student with primary key: 1*

```
Student mystudent = entityManager.find(Student.class, 1);
```

Entity Class

Primary Key

ถ้าไม่เจอมันจะ  
Return เป็น Null  
ออกมา



# การเรียกข้อมูล Java Object

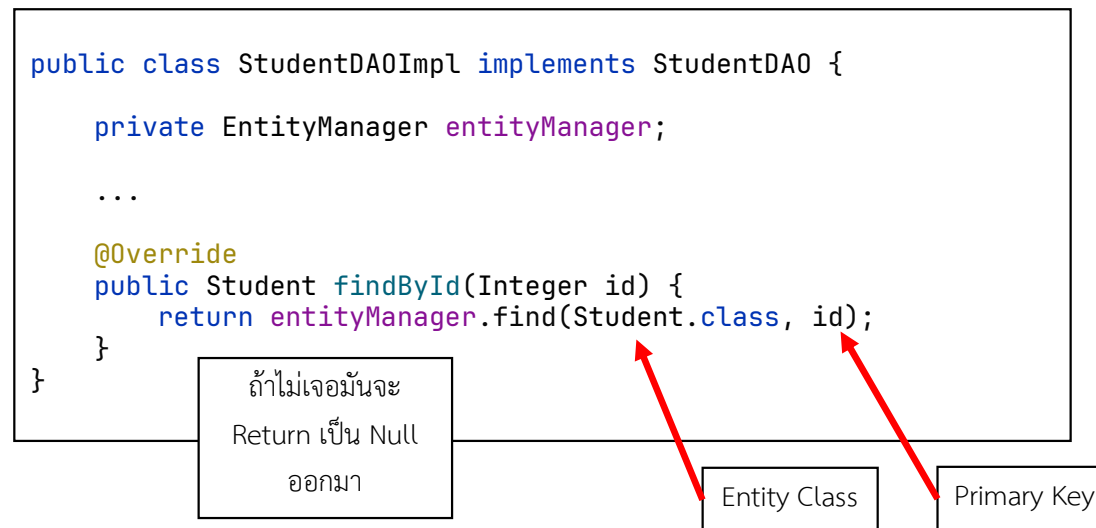
วิธีการทำ

1. เพิ่ม Method ไปที่ DAO interface
2. ไป Implement ตัว Method นี้ที่ DAO implementation
3. ไปใช้ใน Class ที่เราต้องการ

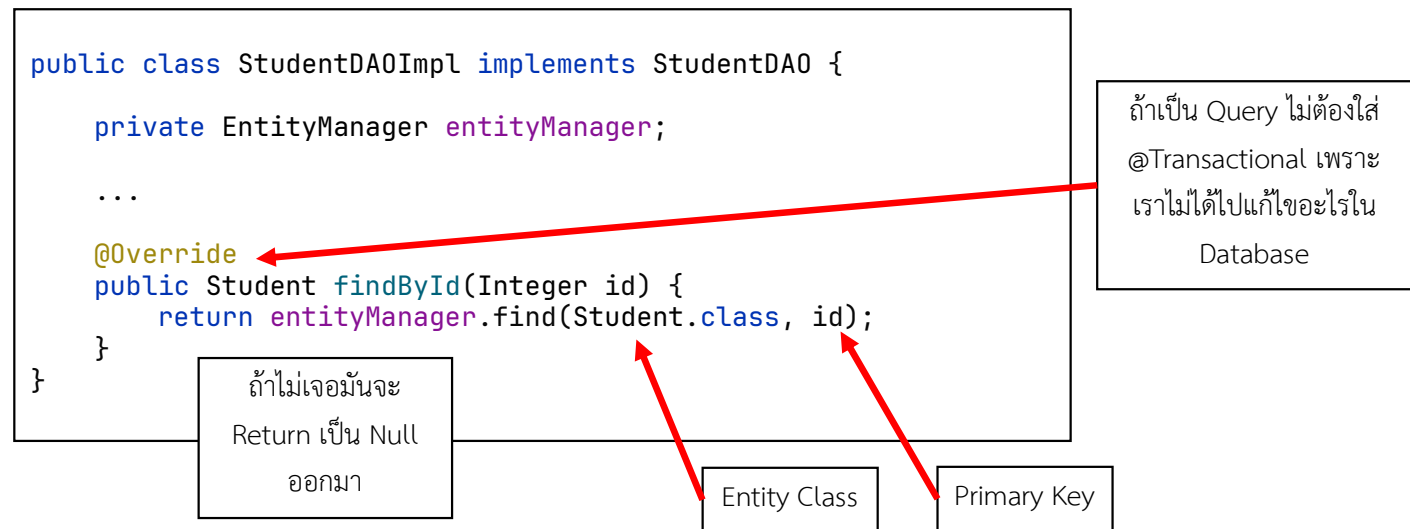
# 1. เพิ่ม Method ไปที่ DAO interface

```
public interface StudentDAO {  
    ...  
    Student findById(Integer id);  
}
```

## 2. ไป Implement ตัว Method นี้ที่ DAO implementation



## 2. ไป Implement ตัว Method นี้ที่ DAO implementation



### 3. ไปใช้ใน Class ที่เราต้องการ

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
        return runner -> {
            readStudent(studentDAO);
        };
    }

    ...
}
```

### 3. ไปใช้ใน Class ที่เราต้องการ

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
        return runner -> {
            readStudent(studentDAO);
        };
    }

    ...
}
```

```
private void readStudent(StudentDAO studentDAO) {
    // create a student object
    System.out.println("Creating new student object...");
    Student tempStudent = new Student("Hello", "World", "hello.w@hotmail.com");

    // save the student object
    System.out.println("Saving the student...");
    studentDAO.save(tempStudent);

    // display id of the saved student
    System.out.println("Saved student. Generated id: " + tempStudent.getId());

    // retrieve student based on the id: primary key
    System.out.println("\nRetrieving student with id: " + tempStudent.getId());
    Student myStudent = studentDAO.findById(tempStudent.getId());
    System.out.println("Found the student: " + myStudent);
}
```

# การเรียกข้อมูล Java Object ด้วย JPA

Mini-workshop: การเรียกข้อมูล Java Object ด้วย JPA

Ref: lab-s4-l5

# JPA/Hibernate CRUD

- Create object
- Read object
- Uppdate object
- Delete object





# JPA Query Language (JPQL)

- เป็น Query Language ใช้สำหรับเรียก Object ออกมาดู
- Concept คล้าย ๆ กับ SQL
  - where, like, order by, join, in, อื่น ๆ
- แต่ JPQL จะใช้ entity name กับ entity fields
  - ไม่ได้ใช้ ชื่อ table กับชื่อคอลัมน์

# Retrieving all Students

```
TypedQuery<Student> theQuery = entityManager.createQuery("FROM Student", Student.class);  
List<Student> students = theQuery.getResultList();
```

เป็นชื่อของ JPA Entity  
(ชื่อของ Entity class)

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...

# Retrieving all Students

```
TypedQuery<Student> theQuery = entityManager.createQuery("FROM Student", Student.class);  
List<Student> students = theQuery.getResultList();
```

เป็นชื่อของ JPA Entity  
(ชื่อของ Entity class)

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...

ไม่ใช่ชื่อของตารางใน Database นะ

ใน JPQL จะใช้ entity name กับ entity  
fields ทั้งหมด

# Retrieving Students

- โดยใช้เงื่อนไข `lastName = 'rukdee'`

```
TypedQuery<Student> theQuery = entityManager.createQuery(  
    "FROM Student WHERE lastName='rukdee'", Student.class);  
  
List<Student> students = theQuery.getResultList();
```

เป็นชื่อของ Field ใน JPA  
Entity

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...

# Retrieving Students

- โดยใช้เงื่อนไข OR predicate

เป็นชื่อของ Field ใน JPA  
Entity

```
TypedQuery<Student> theQuery = entityManager.createQuery(
    "FROM Student WHERE lastName='rukdee' OR firstName='somchai'", Student.class);
List<Student> students = theQuery.getResultList();
```

## Java Class

### Student

- id: int
- firstName : String
- lastName : String
- email : String

...

# Retrieving Students

- โดยใช้เงื่อนไข OR predicate

```
TypedQuery<Student> theQuery = entityManager.createQuery(  
    "FROM Student WHERE lastName='rukdee' OR firstName='somchai'", Student.class);  
List<Student> students = theQuery.getResultList();
```

เป็นชื่อของ Field ใน JPA  
Entity

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...

# Retrieving Students

- โดยใช้เงื่อนไข OR predicate

```
TypedQuery<Student> theQuery = entityManager.createQuery(  
    "FROM Student WHERE lastName='rukdee' OR firstName='somchai'", Student.class);  
List<Student> students = theQuery.getResultList();
```

เป็นชื่อของ Field ใน JPA  
Entity

Java Class

Student

- id: int  
- firstName : String  
- lastName : String  
- email : String

...

เป็นชื่อของ Field ใน JPA  
Entity

# Retrieving Students

- โดยใช้เงื่อนไข LIKE predicate

```
TypedQuery<Student> theQuery = entityManager.createQuery(  
    "FROM Student WHERE email like '%hotmail.com'", Student.class);  
  
List<Student> students = theQuery.getResultList();
```

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...



# JPQL – Named Parameters

- ใช้ Named Parameters

JPQL Named Parameters จะขึ้นต้นด้วย colon:

```
public List<Student> findByLastName(String theLastName) {  
    TypedQuery<Student> theQuery = entityManager.createQuery(  
        "FROM Student WHERE lastName=:theData", Student.class);  
    theQuery.setParameter("theData", theLastName);  
    return theQuery.getResultList();  
}
```

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...

อาจจะมองว่าเป็นช่องที่เว้นว่างไว้เติมทีหลังก็ได้

# JPQL – Named Parameters

- ใช้ Named Parameters

JPQL Named Parameters จะขึ้นต้น  
ด้วย colon:

```
public List<Student> findByLastName(String theLastName) {  
    TypedQuery<Student> theQuery = entityManager.createQuery(  
        "FROM Student WHERE lastName=:theData", Student.class);  
    theQuery.setParameter("theData", theLastName);  
    return theQuery.getResultList();  
}
```

Java Class

Student

- id: int
- firstName : String
- lastName : String
- email : String

...

อาจจะมองว่าเป็นช่องที่เว้นว่างไว้เติมที่  
หลังก็ได้

# Query with JPQL

วิธีการทำ

1. เพิ่ม Method ไปที่ DAO interface
2. ไป Implement ตัว Method นี้ที่ DAO implementation
3. ไปใช้ใน Class ที่เราต้องการ

# 1. เพิ่ม Method ไปที่ DAO interface

```
public interface StudentDAO {  
    ...  
    List<Student> findAll();  
}
```

## 2. ไป Implement ตัว Method นี้ที่ DAO implementation

```
public class StudentDAOImpl implements StudentDAO {  
    private EntityManager entityManager;  
    ...  
    @Override  
    public List<Student> findAll() {  
        TypedQuery<Student> theQuery = entityManager.createQuery("FROM Student", Student.class);  
        return theQuery.getResultList();  
    }  
}
```

ไม่ต้องใส่ @Transactional  
ถ้าเราแค่ Query

เป็นชื่อของ JPA Entity

### 3. ไปใช้ใน Class ที่เราต้องการ

```
@SpringBootApplication
public class CruddemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(CruddemoApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
        return runner -> {
            queryForStudents(studentDAO);
        };
    }

    private void queryForStudents(StudentDAO studentDAO) {
        // get list of students
        List<Student> students = studentDAO.findAll();

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

# Query ด้วย JPQL

Mini-workshop: Query ด้วย JPQL

Ref: lab-s4-l6

# QUERY ด้วย JPQL



(Homework-E1) Mini-workshop: Query ด้วย JPQL

- ใช้ endpoint จาก lab-s4-l6 โดยให้สามารถ ค้นหาจาก lastname ผ่าน query parameter
- เช่น <http://localhost:8080/find-by-lastname?q=potter>



# JPA/Hibernate CRUD

- Create object
- Read object
- Uppdate object
- Deleate object




# เปลี่ยนแปลงค่า Java Object

- อัปเดตตัวเดียว

```
Student student = entityManager.find(Student.class, 1);  
  
// change first name to "Somying"  
theStudent.setFirstName("Somying");  
  
entityManager.merge(student);
```

Update ตัว Entity



# เปลี่ยนแปลงค่า Java Object

- เปลี่ยนแปลงค่า **lastname** ของ **Students** ทั้งหมด (อัปเดตหลายตัว)

## Java Class

Student
- id: int - firstName : String - lastName : String - email : String
...

ชื่อของ JPA Entity (ชื่อ  
ของ Entity class)

```
int numRowsUpdated = entityManager.createQuery("UPDATE Student SET lastName='Tester'")  
    .executeUpdate();
```

# เปลี่ยนแปลงค่า Java Object

- เปลี่ยนแปลงค่า **lastname** ของ **Students** ทั้งหมด (อัปเดตหลายตัว)

## Java Class

Student
- id: int - firstName : String - <b>lastName</b> : String - email : String
...

```
int numRowsUpdated = entityManager.createQuery("UPDATE Student SET lastName='Tester'")  
    .executeUpdate();
```

ชื่อของ JPA Entity (ชื่อ  
ของ Entity class)

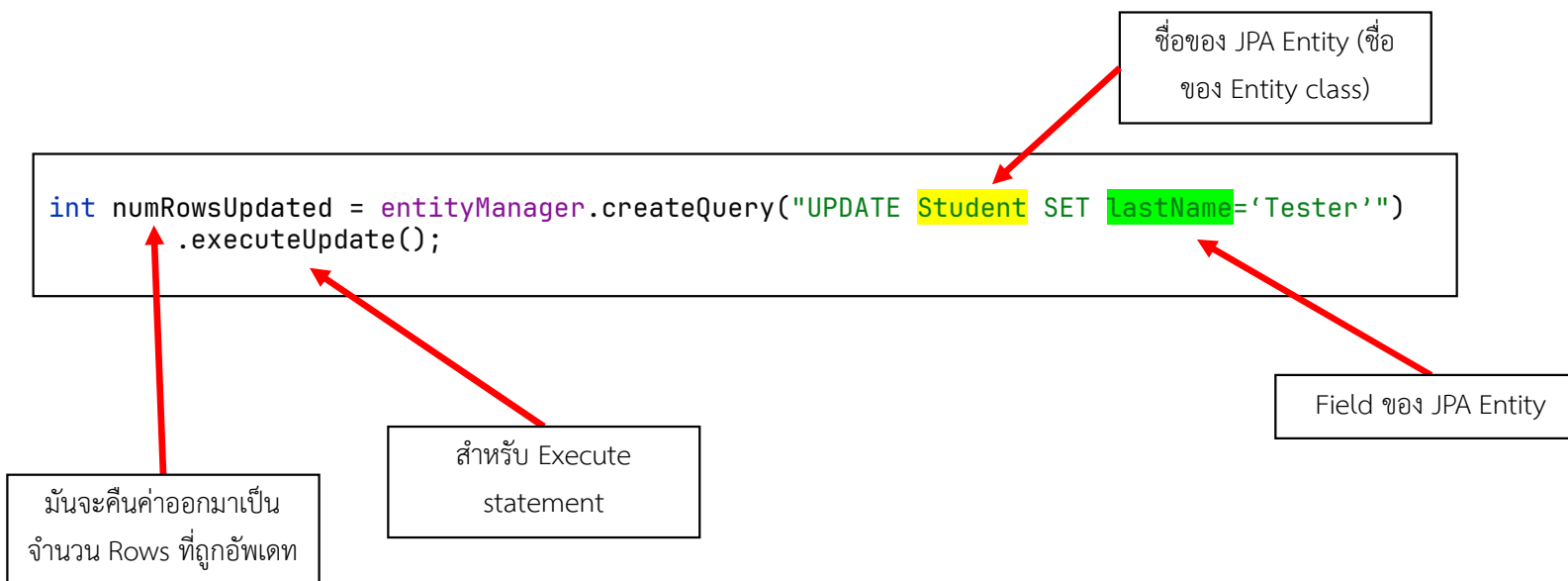
Field ของ JPA Entity

# เปลี่ยนแปลงค่า Java Object

- เปลี่ยนแปลงค่า lastname ของ Students ทั้งหมด (อัปเดตหลายตัว)

## Java Class

Student
<ul style="list-style-type: none"><li>- id: int</li><li>- firstName : String</li><li>- lastName : String</li><li>- email : String</li></ul>
...




# เปลี่ยนแปลงค่า Java Object

วิธีการทำ

1. เพิ่ม Method ไปที่ DAO interface
2. ไป Implement ตัว Method นี้ที่ DAO implementation
3. ไปใช้ใน Class ที่เราต้องการ

# 1. เพิ่ม Method ไปที่ DAO interface

```
public interface StudentDAO {  
    ...  
    void update(Student student);  
}
```



## 2. ไป Implement ตัว Method นี้ที่ DAO implementation

```
public class StudentDAOImpl implements StudentDAO {  
    private EntityManager entityManager;  
  
    ...  
  
    @Override  
    @Transactional  
    public void update(Student theStudent) {  
        entityManager.merge(theStudent);  
    }  
}
```

ต้องใส่ @Transactional  
เพราะมีการเปลี่ยนแปลง  
ข้อมูลใน Database



### 3. ไปใช้ใน Class ที่เราต้องการ

```
@SpringBootApplication
public class FirstSpringBootApplication {
    ...

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
        return runner -> {
            updateStudent(studentDAO);
        };
    }
}
```

```
private void updateStudent(StudentDAO studentDAO) {

    // retrieve student based on the id: primary key
    int studentId = 1;
    System.out.println("Getting student with id: " + studentId);

    Student myStudent = studentDAO.findById(studentId);

    System.out.println("Updating student...");

    // change first name to "Scooby"
    myStudent.setFirstName("Scooby");
    studentDAO.update(myStudent);

    // display updated student
    System.out.println("Updated student: " + myStudent);
}
```

# เปลี่ยนแปลงค่า Java Object ด้วย JPA

Mini-workshop: เปลี่ยนแปลงค่า Java Object ด้วย JPA

Ref: lab-s4-l7

# เปลี่ยนแปลงค่า JAVA OBJECT ด้วย JPA

(Homework) Mini-workshop: เปลี่ยนแปลงค่า Java Object ด้วย JPA

- ให้ทำต่อจาก lab-s4-l7 โดยเพิ่ม endpoint ให้ทำให้สามารถ update จาก id และ firstname ที่ต้องการผ่าน api ได้
- เช่น <http://localhost:8080/update-student-firstname?id=1&firstname=somchai>
- ตัวอย่างด้านบนจะเปลี่ยน firstname ของ student ที่มี id = 1

# เปลี่ยนแปลงค่า JAVA OBJECT ด้วย JPA

(Homework-H1) Mini-workshop: เปลี่ยนแปลงค่า Java Object ด้วย JPA

- ให้ทำต่อจาก lab-s4-l7 โดยเพิ่ม endpoint ให้ทำให้สามารถ update firstname lastname หรือ email จาก id ผ่าน api ได้
- เช่น <http://localhost:8080/update-student-data?id=1&firstname=somchai>
- ตัวอย่างด้านบนจะเปลี่ยน firstname ของ student ที่มี id = 1
- <http://localhost:8080/update-student-data?id=1&lastname=tumdee>
- ตัวอย่างด้านบนจะเปลี่ยน lastname ของ student ที่มี id = 1
- <http://localhost:8080/update-student-data?id=1&email=tumdee@hotmail.com>
- ตัวอย่างด้านบนจะเปลี่ยน email ของ student ที่มี id = 1

# เปลี่ยนแปลงค่า JAVA OBJECT ด้วย JPA

(Homework-H2) Mini-workshop: เปลี่ยนแปลงค่า Java Object ด้วย JPA

- ให้ทำต่อจาก lab-s4-l7 โดยเพิ่ม endpoint ให้ทำให้สามารถ update firstname lastname หรือ email พร้อมกัน ได้ จาก id ผ่าน api ได้
- เช่น <http://localhost:8080/update-student-data?id=1&firstname=somchai&lastname=tumdee>
- ตัวอย่างด้านบนจะเปลี่ยน firstname และ lastname ของ student ที่มี id = 1
- <http://localhost:8080/update-student-data?id=1&lastname=tumdee&email=tumdee@hotmail.com>
- ตัวอย่างด้านบนจะเปลี่ยน firstname lastname และ email ของ student ที่มี id = 1
- <http://localhost:8080/update-student-data?id=1&email=tumdee@hotmail.com>
- ตัวอย่างด้านบนจะเปลี่ยน email ของ student ที่มี id = 1

# JPA/Hibernate CRUD

- Create object
- Read object
- Uppdate object
- Delete object



# ลบ Java Object

- ลบ Student อันหนึ่ง

```
// retrieve the student  
int id = 1;  
Student student = entityManager.find(Student.class, id);  
  
// delete the student  
entityManager.remove(student);
```

# ลบ Java Object

- ลบ Student โดยใช้เงื่อนไข

Java Class

**Student**

- id: int
- firstName : String
- lastName : String
- email : String

ชื่อของ JPA Entity (ชื่อ  
ของ Entity class)

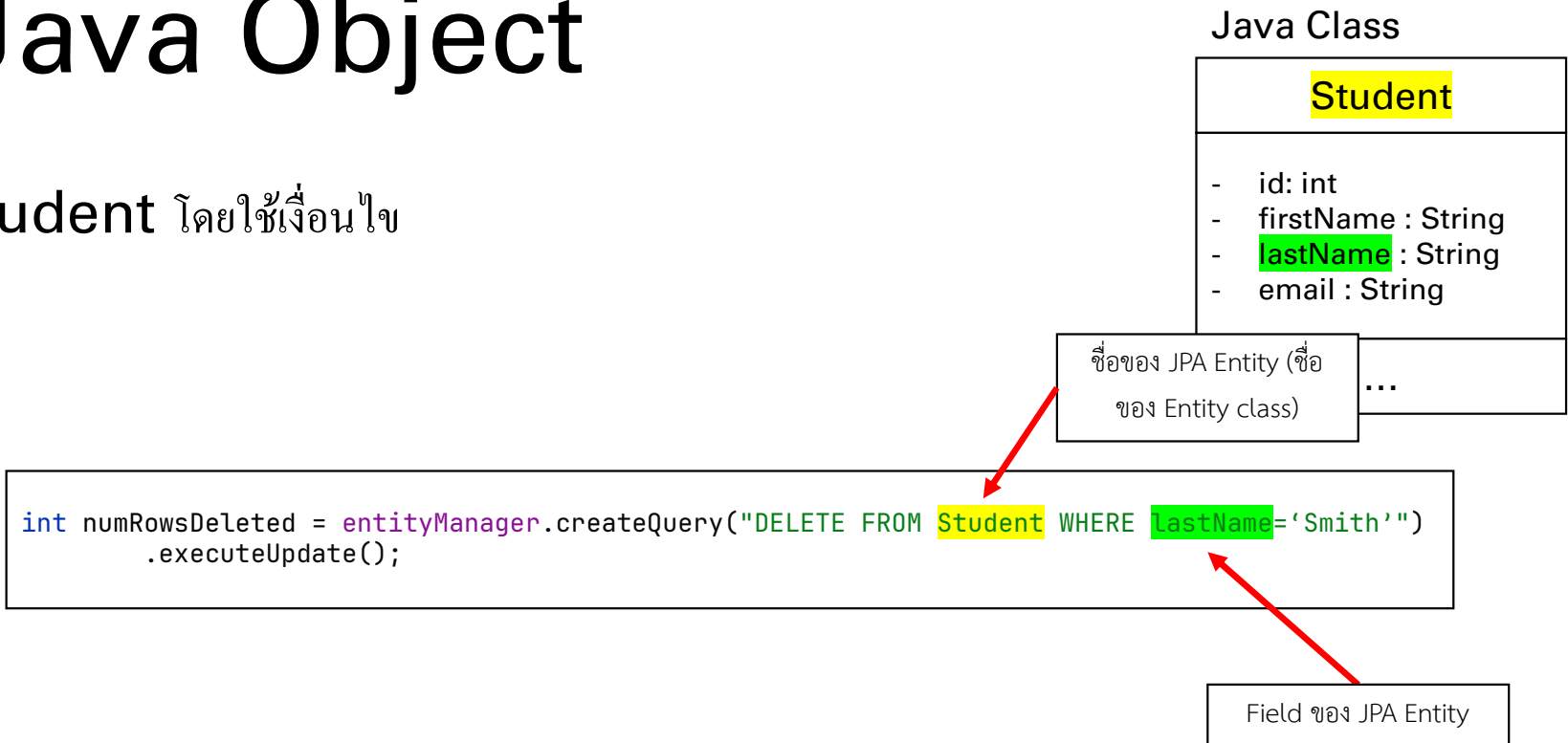
...

```
int numRowsDeleted = entityManager.createQuery("DELETE FROM Student WHERE lastName='Smith'")  
    .executeUpdate();
```



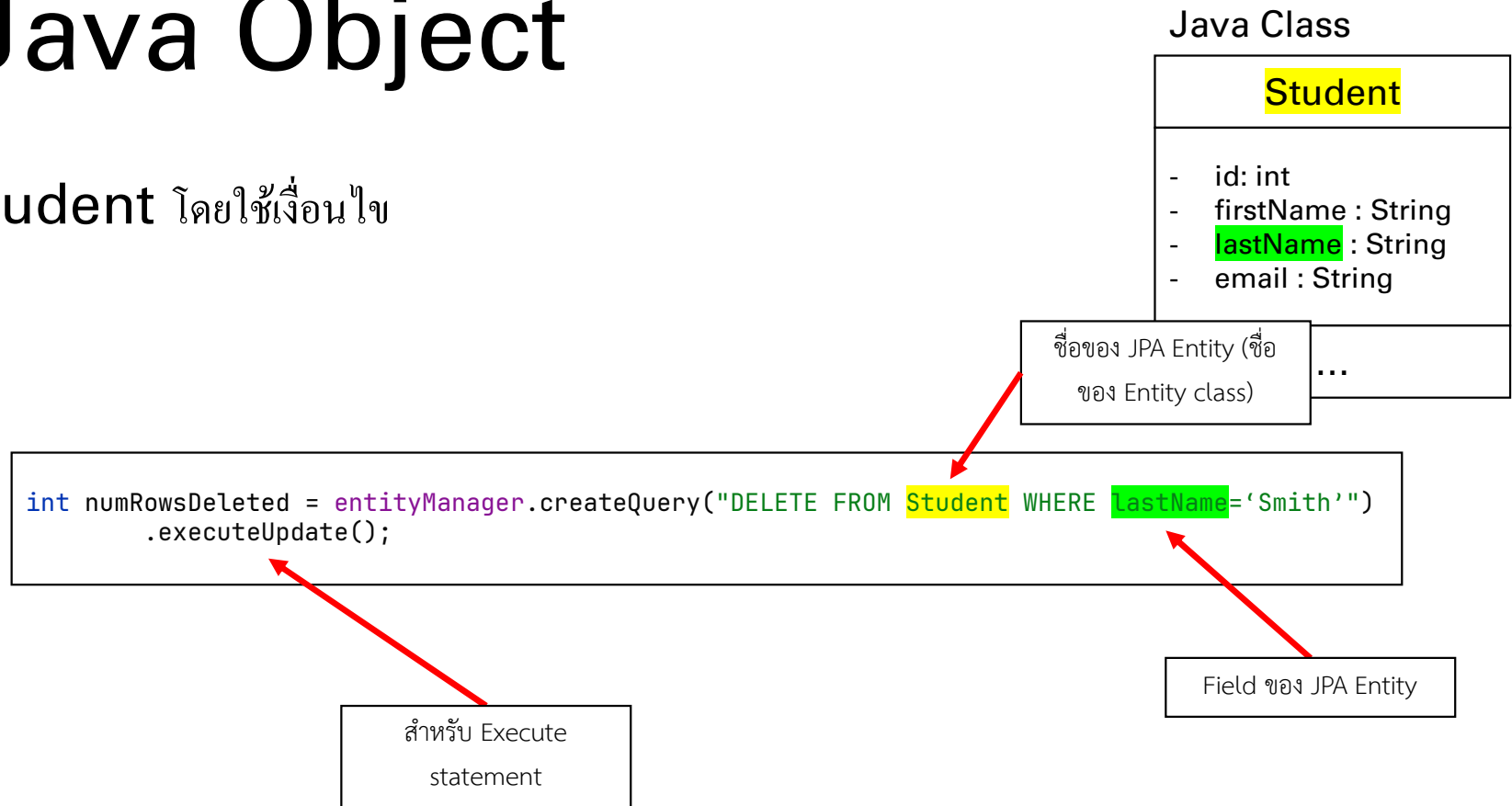
# ลบ Java Object

- ลบ Student โดยใช้เงื่อนไข



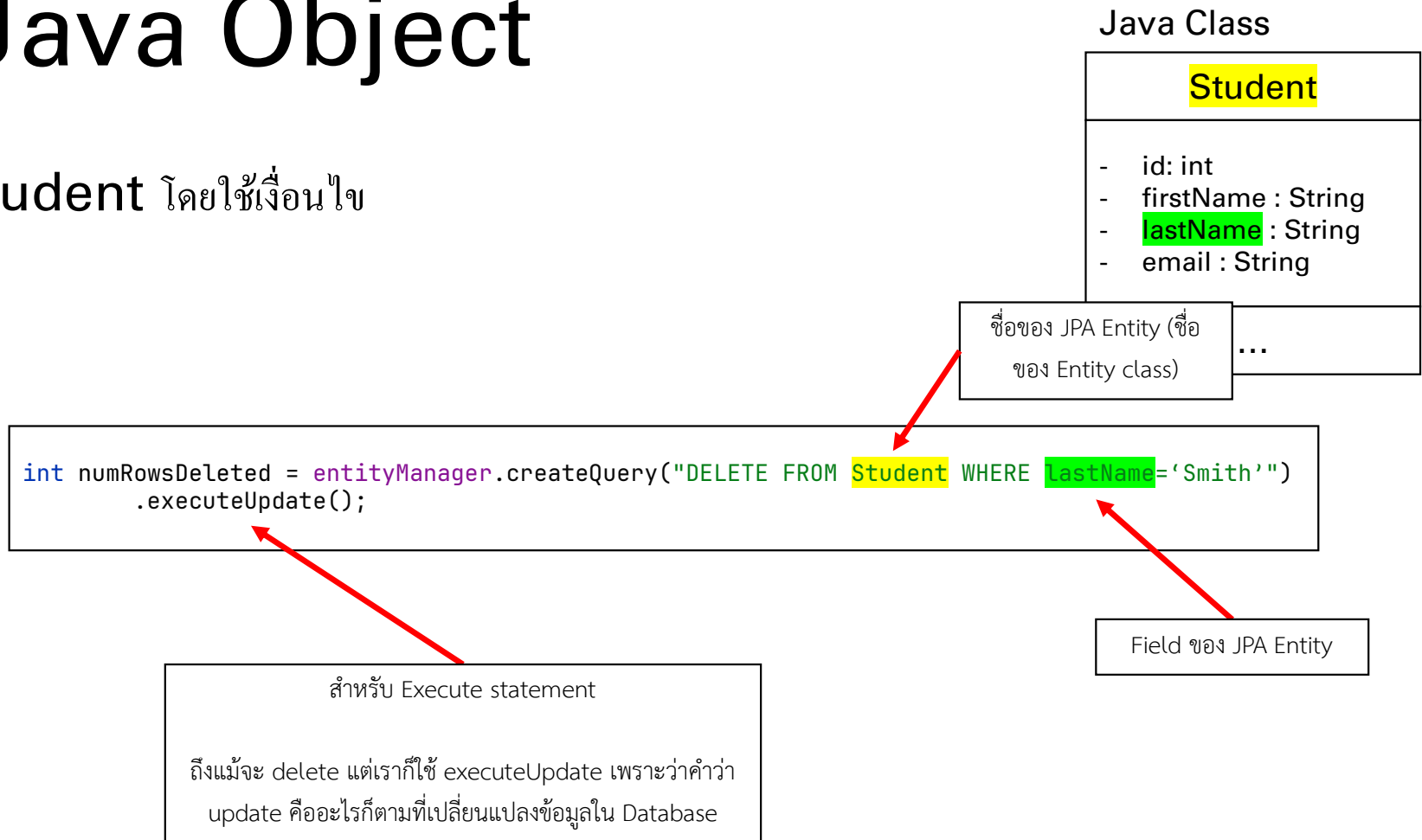
# ลบ Java Object

- ลบ Student โดยใช้เงื่อนไข



# ลบ Java Object

- ลบ Student โดยใช้เงื่อนไข



# ลบ Java Object

- ลบ Student โดยใช้เงื่อนไข

## Java Class

### Student

- id: int
- firstName : String
- lastName : String
- email : String

ชื่อของ JPA Entity (ชื่อ  
ของ Entity class)

...

```
int numRowsDeleted = entityManager.createQuery("DELETE FROM Student WHERE LastName='Smith'")  
    .executeUpdate();
```

Field ของ JPA Entity

สำหรับ Execute statement

มันจะคืนค่าออกมาเป็น  
จำนวน Rows ที่ถูกลบ

ถึงแม้จะ delete แต่เราก็ใช้ executeUpdate เพราะว่าคำว่า  
update คืออะไรก็ตามที่เปลี่ยนแปลงข้อมูลใน Database

# ลบ Java Object

- ลบ Students ทั้งหมด

```
int numRowsDeleted = entityManager
    .createQuery("DELETE FROM Student")
    .executeUpdate();
```


# ลบ Java Object

วิธีการทำ

1. เพิ่ม Method ไปที่ DAO interface
2. ไป Implement ตัว Method นี้ที่ DAO implementation
3. ไปใช้ใน Class ที่เราต้องการ

# 1. เพิ่ม Method ไปที่ DAO interface

```
public interface StudentDAO {  
    ...  
    void delete(Integer id);  
}
```



## 2. ไป Implement ตัว Method นี้ที่ DAO implementation

```
public class StudentDAOImpl implements StudentDAO {  
    private EntityManager entityManager;  
  
    ...  
  
    @Override  
    @Transactional  
    public void delete(Integer id) {  
        Student theStudent = entityManager.find(Student.class, id);  
        entityManager.remove(theStudent);  
    }  
}
```

ต้องใส่ @Transactional  
เพราะว่ามีการเปลี่ยนแปลง  
Database (ลบ Rows)



### 3. ไปใช้ใน Class ที่เราต้องการ

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
        return runner -> {
            deleteStudent(studentDAO);
        };
    }

    private void deleteStudent(StudentDAO studentDAO) {
        // delete the student
        int studentId = 3;

        System.out.println("Deleting student id: " + studentId);

        studentDAO.delete(studentId);
    }
}
```

# ลบ Java Object ด้วย JPA

Mini-workshop: ลบ Java Object ด้วย JPA

Ref: lab-s4-l8

# ลบ JAVA OBJECT ด้วย JPA

(Homework-E2) Mini-workshop: ลบ Java Object ด้วย JPA

- ให้ทำต่อจาก lab-s4-l8 โดยเพิ่ม endpoint ให้ทำให้สามารถ delete โดยใช้ id ได้
- เช่น <http://localhost:8080/delete-by-student-id?id=2>
- ตัวอย่างข้างบนจะทำการลบ Student ที่มี id = 2