Java 2

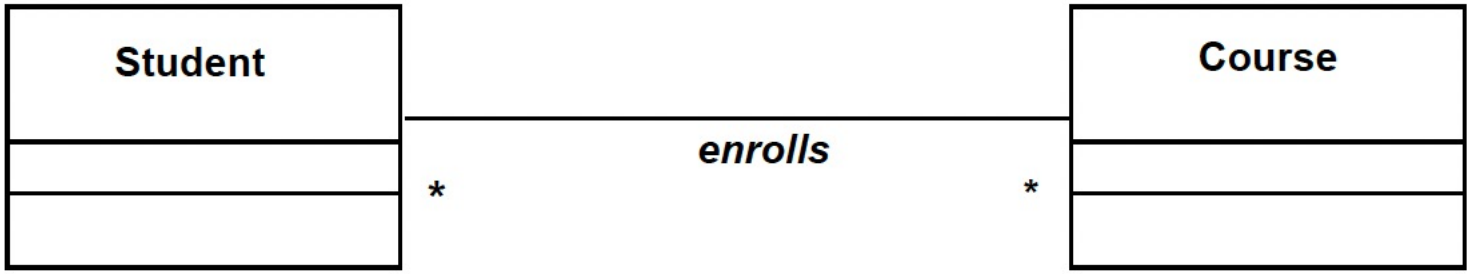
Hibernate Annotation

with

Class Relationship II

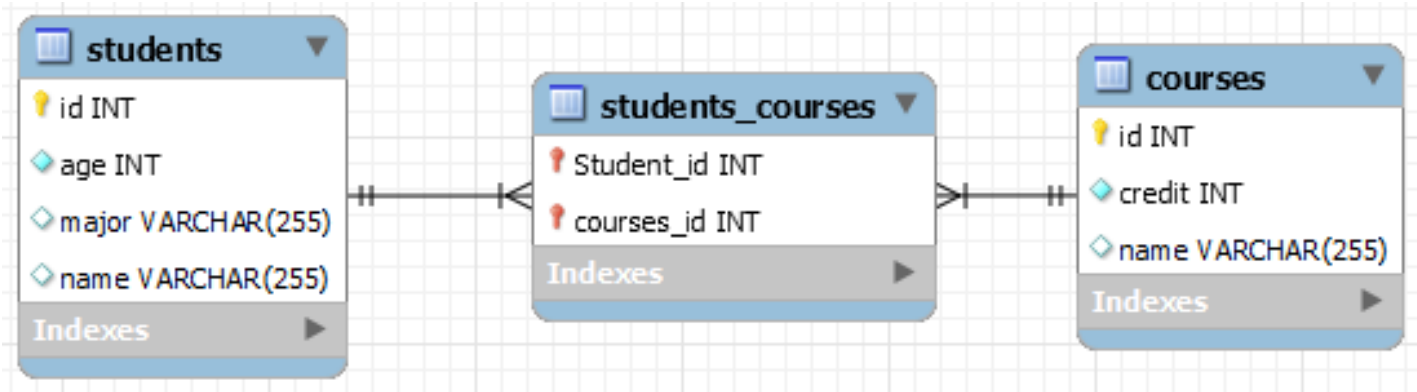
Many To Many

- ความสัมพันธ์แบบกลุ่มต่อกลุ่มเกิดขึ้นเมื่ออ็อบเจกต์ X มีการอ้างอิงไปยังกลุ่มของอ็อบเจกต์ Y ในขณะที่อ็อบเจกต์ Y มีการอ้างอิงไปยังกลุ่มของอ็อบเจกต์ X ในลักษณะที่เป็นความสัมพันธ์แบบกลุ่มต่อกลุ่ม
- ตัวอย่างเช่น Student แต่ละคนสามารถลงทะเบียนเรียนได้หลาย Course และแต่ละ Course ประกอบไปด้วย Student ที่ลงทะเบียนเรียนหลายคน



Many To Many : DB

- Association Class เมื่อแปลงให้เป็นตารางฐานข้อมูลต้องสร้างความสัมพันธ์ผ่านตารางที่สามเพื่อเชื่อมความสัมพันธ์ระหว่างกัน ได้แก่ ตาราง Student ที่มีความสัมพันธ์แบบ 1: M ไปยังตาราง Student_Course และตาราง Course มีความสัมพันธ์แบบ 1: M ไปยังตาราง Student_Course ดังรูป



Many To Many : Annotation

- สัญลักษณ์ @ManyToManyใช้ระบุความสัมพันธ์แบบกลุ่มต่อกลุ่ม
- สัญลักษณ์ @ManyToMany โดยปกติมักใช้ร่วมกับพารามิเตอร์ cascade เพื่อให้การบันทึกข้อมูลอ็อบเจกต์ต่อเนื่อง เช่น บันทึก Student อ็อบเจกต์เดียวแต่ส่งผลต่อเนื่องไปยังการบันทึก Course อ็อบเจกต์โดยอัตโนมัติ
- เนื่องจากเป็นแบบทิศทางเดียว @ManyToMany จะถูกกำหนดไว้ที่เอนติทีที่เป็น Owner เสมอ ในกรณีนี้ได้แก่ Student

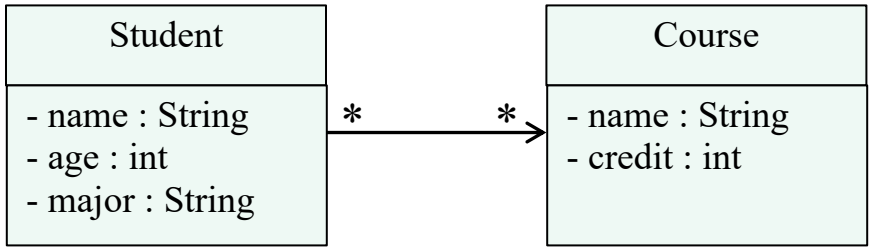
@ManyToMany(cascade = CascadeType.ALL)

1. Uni-directional Many To Many

Student and Course classes

```
public class Student {
  private int id;
  private String name;
  private int age;
  private String major;
  private Set<Course> courses = new HashSet<Course>();

  ...
}
```

```
public class Course {
  private int id;
  private String name;
  private int credit;

  ...
}
```



Uni-directional Many To Many

```
@Entity
@Table(name="students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
  private int age;
  private String major;

  @ManyToMany(cascade = {CascadeType.ALL})
  private Set<Course> courses = new HashSet<Course>();

  ...
}
```

```
@Entity
@Table(name="courses")
public class Course {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
  private int credit;

  ...;
}
```

Uni-Directional ไม่มีการประกาศตัวแปร Student Set

Run

```
Student student1 = new Student("Somchai", 22, "Information Technology");
Student student2 = new Student("Somsri", 23, "Computer Science");
Course course1 = new Course("Java", 3);
Course course2 = new Course("Uml", 3);
student1.getCourses().add(course1);
student1.getCourses().add(course2);
student2.getCourses().add(course1);
student2.getCourses().add(course2);
session.save(student1);
session.save(student2);
```

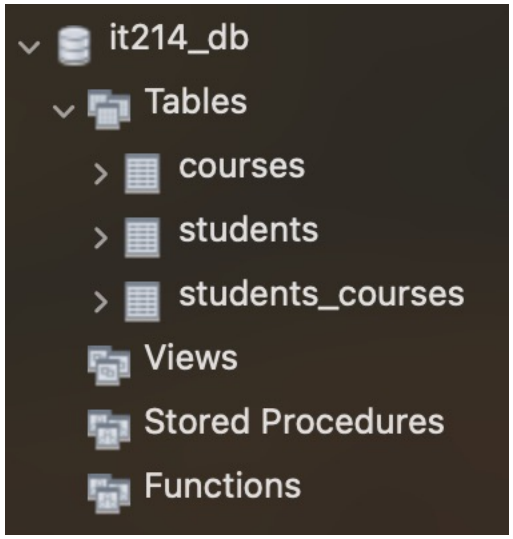


Table: students

id	age	major	name
1	22	Information Technology	Somchai
2	23	Computer Science	Somsri

Table: courses

id	credit	name
1	3	Uml
2	3	Java

Table: students_courses

Student_id	courses_id
1	1
1	2
2	1
2	2

ชื่อคลาสแรกจะขึ้นต้นด้วยตัวอักษรภาษาอังกฤษตัวใหญ่

List all Student and Course Relationship


```
Student student1 = session.get(Student.class, 1);
System.out.println("Student's ID: " + student1.getId());
System.out.println("Student's Name: " + student1.getName());
System.out.println("Student's Age: " + student1.getAge());
Set<Course> courses = student1.getCourses();
for (Course course : courses) {
  System.out.println(" Course's ID: " + course.getId() + ", "
    + "Course's Name: " + course.getName());
}

Student student2 = session.get(Student.class, 2);
System.out.println("Student's ID: " + student2.getId());
System.out.println("Student's Name: " + student2.getName());
System.out.println("Student's Age: " + student2.getAge());
courses = student2.getCourses();
for (Course course : courses) {
  System.out.println(" Course's ID: " + course.getId() + ", "
    + "Course's Name: " + course.getName());
}
```

Result


```
Student's ID: 1
Student's Name: Somchai
Student's Age: 22
  Course's ID: 1, Course's Name: Java
  Course's ID: 2, Course's Name: Uml
Student's ID: 2
Student's Name: Somsri
Student's Age: 23
  Course's ID: 1, Course's Name: Java
  Course's ID: 2, Course's Name: Uml
```

สัญลักษณ์ @JoinTable

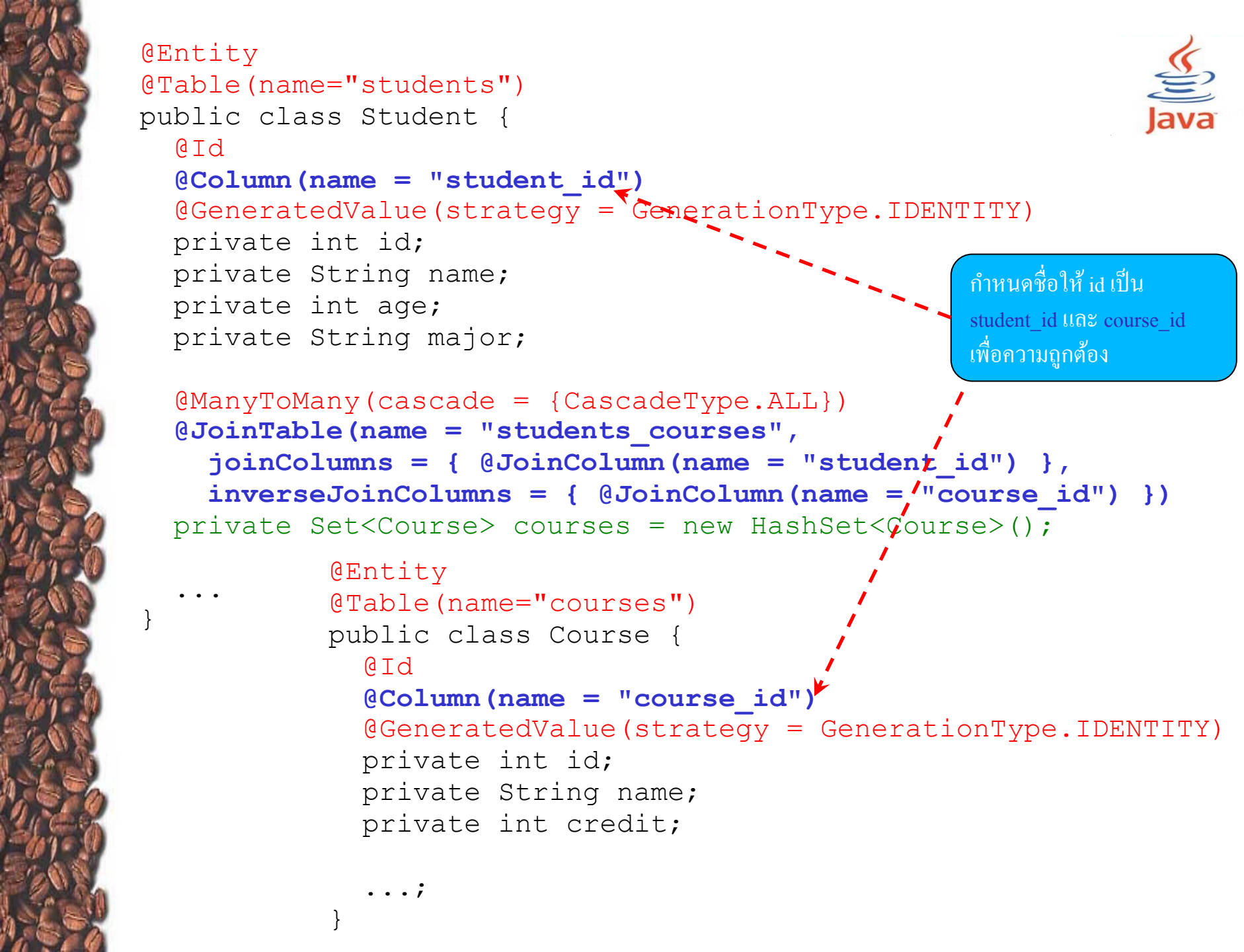
- ในกรณีที่เราใช้ Hibernate กับระบบฐานข้อมูลเดิมซึ่งมีความสัมพันธ์ Many-To-Many และมีตารางฐานข้อมูลเดิมอยู่แล้วเราสามารถกำหนดให้ Hibernate เชื่อมตารางแบบ Many-To-Many ได้ด้วย @JoinTable
- จากตัวอย่าง ความสัมพันธ์แบบกลุ่มต่อกลุ่มระหว่างเอนติที Student และ Course มีการสร้างเอนติทีที่สาม ได้คือ students_courses สามารถทำได้ดังนี้
  - กำหน Foreign Key ตัวแรกเพื่อเชื่อมโยงความสัมพันธ์ผ่านไปยัง Primary Key ของเอนติที Student ที่อยู่ต้นทาง (joinColumns)
  - กำหนด Foreign Key ตัวที่สองเพื่อเชื่อมโยงความสัมพันธ์ผ่านไปยัง Primary Key ของเอนติที Course ที่อยู่ปลายทาง (inverseJoinColumns)
  - ปกติแล้ว Primary Key แบบผสมของตารางที่สามเกิดจากการรวมตัวของ Foreign Key ของทั้งสองตาราง

**การใช้ @JoinTable เป็นทางเลือกจะใช้เมื่อมีฐานข้อมูลเดิมอยู่แล้วและนำ Hibernate มาใช้ในภายหลัง หรือเมื่อต้องการปรับแต่รายละเอียดต่าง ๆ ในการออกแบบเอง

สัญลักษณ์ @JoinTable

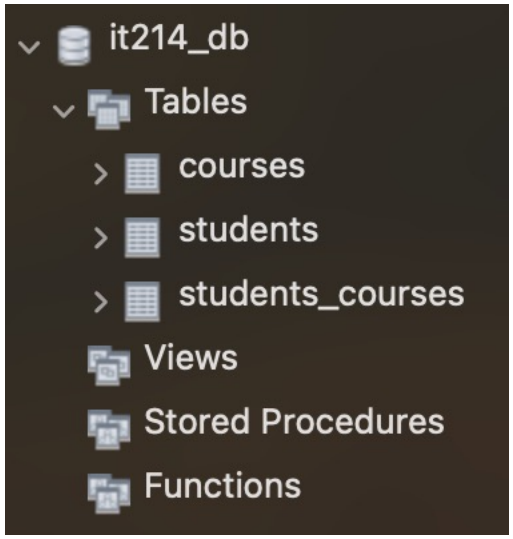
- สัญลักษณ์ @JoinTable ประกอบด้วยค่าพารามิเตอร์ต่าง ๆ ดังนี้
  - Name ชื่อของตารางที่สามที่ถูกสร้างขึ้น
  - JoinColumns ใช้กำหนดชื่อคอลัมน์ในตารางที่สามเพื่อเชื่อมโยงไปยังสองตารางแรก และใช้ร่วมกับ referencedColumnName ที่ประกอบด้วยชื่อคอลัมน์ Primary Key ของด้านที่เป็น Owner
  - inverseJoinColumns ทำหน้าที่รับผิดชอบเกี่ยวกับการแปลงคอลัมน์ทางด้าน inverse ตรงกันข้ามกับด้าน Qwner ตัวอย่างเช่น

```
@JoinTable(name = "students_courses",
        joinColumns = { @JoinColumn(name = "student_id")},
        inverseJoinColumns = { @JoinColumn(name = "course_id")
})
```


@Entity
@Table(name="students")
public class Student {
@Id
@Column(name = "student_id")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private String name;
private int age;
private String major;
@ManyToMany(cascade = {CascadeType.ALL})
@JoinTable(name = "students_courses",
joinColumns = { @JoinColumn(name = "student_id") },
inverseJoinColumns = { @JoinColumn(name = "course_id") })
private Set<Course> courses = new HashSet<Course>();
...
}
@Entity
@Table(name="courses")
public class Course {
@Id
@Column(name = "course_id")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private String name;
private int credit;
...;
}
กำหนดชื่อให้ id เป็น
student_id และ course_id
เพื่อความถูกต้อง

Run

```
Student student1 = new Student("Somchai", 22, "Information Technology");
Student student2 = new Student("Somsri", 23, "Computer Science");
Course course1 = new Course("Java", 3);
Course course2 = new Course("Uml", 3);
student1.getCourses().add(course1);
student1.getCourses().add(course2);
student2.getCourses().add(course1);
student2.getCourses().add(course2);
session.save(student1);
session.save(student2);
```



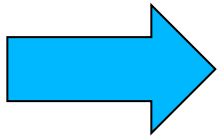


Table: students

student_id	age	major	name
1	22	Information Technology	Somchai
2	23	Computer Science	Somsri

Table: courses

course_id	credit	name
1	3	Java
2	3	Uml

Table: students_courses

student_id	course_id
1	1
1	2
2	1
2	2

List all Student and Course Relationship


```
Student student1 = session.get(Student.class, 1);
System.out.println("Student's ID: " + student1.getId());
System.out.println("Student's Name: " + student1.getName());
System.out.println("Student's Age: " + student1.getAge());
Set<Course> courses = student1.getCourses();
for (Course course : courses) {
  System.out.println(" Course's ID: " + course.getId() + ", "
    + "Course's Name: " + course.getName());
}

Student student2 = session.get(Student.class, 2);
System.out.println("Student's ID: " + student2.getId());
System.out.println("Student's Name: " + student2.getName());
System.out.println("Student's Age: " + student2.getAge());
courses = student2.getCourses();
for (Course course : courses) {
  System.out.println(" Course's ID: " + course.getId() + ", "
    + "Course's Name: " + course.getName());
}
```

Result


```
Student's ID: 1
Student's Name: Somchai
Student's Age: 22
  Course's ID: 1, Course's Name: Java
  Course's ID: 2, Course's Name: Uml
Student's ID: 2
Student's Name: Somsri
Student's Age: 23
  Course's ID: 1, Course's Name: Java
  Course's ID: 2, Course's Name: Uml
```

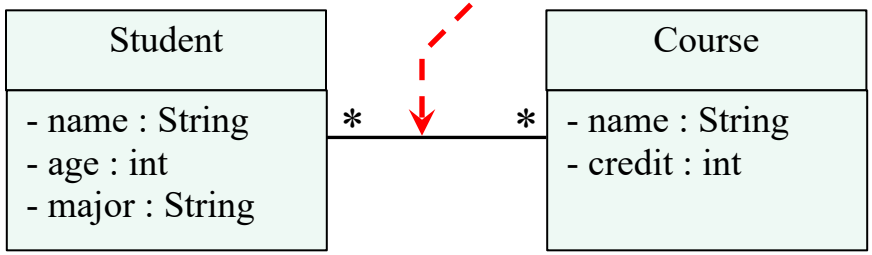
2. Bi-directional Many To Many

Bi-directional Many To Many

```
public class Student {
   private int id;
   private String name;
   private int age;
   private String major;
   private Set<Course> courses = new HashSet<Course>();

   ...
}

public class Course {
   private int id;
   private String name;
   private int credit;
   private Set<Student> students = new HashSet<Student>();

   ...
}
```

ทั้งสองคลาสมี Attribute ของอีกคลาส

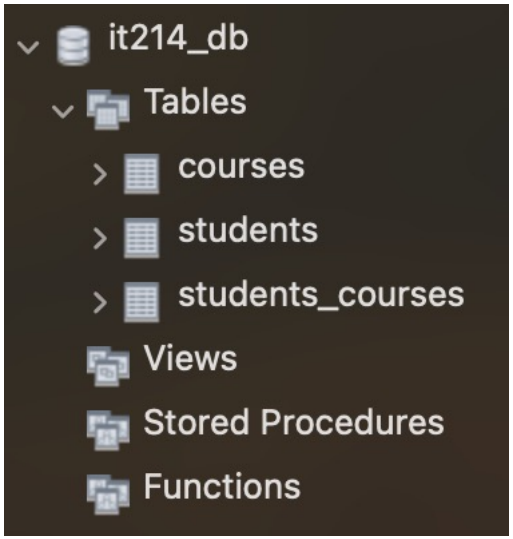
เส้นความสัมพันธ์จะไม่มีหัวลูกศรทั้งสองด้าน



Bi-directional Many To Many


```
@Entity
@Table(name="students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
  private int age;
  private String major;

  @ManyToMany(cascade = {CascadeType.ALL})
  private Set<Course> courses = new HashSet<Course>();

  ...;
}

@Entity
@Table(name="courses")
public class Course {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
  private int credit;
  @ManyToMany(cascade = CascadeType.ALL, mappedBy = "courses")
  private Set<Student> students = new HashSet<Student>();

  ...;
}
```

```
Student student1 = new Student("Somchai", 22, "Information Technology");
Student student2 = new Student("Somsri", 23, "Computer Science");
Course course1 = new Course("Java", 3);
Course course2 = new Course("Uml", 3);

student1.getCourses().add(course1);
student1.getCourses().add(course2);
student2.getCourses().add(course1);
student2.getCourses().add(course2);

course1.getStudents().add(student1);
course1.getStudents().add(student2);
course2.getStudents().add(student1);
course2.getStudents().add(student2);

session.save(student1);
session.save(student2);
```



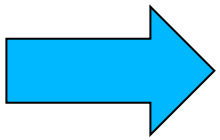


Table: students

id	age	major	name
1	22	Information Technology	Somchai
2	23	Computer Science	Somsri

Table: courses

id	credit	name
1	3	Uml
2	3	Java

Table: students_courses

student_id	course_id
1	1
1	2
2	1
2	2

List all Student and Course Relationship


```
Course course1 = session.get(Course.class, 1);
System.out.println("Course's ID: " + course1.getId());
System.out.println("Course's Name: " + course1.getName());
System.out.println("Course's Credit: " + course1.getCredit());
Set<Student> students = course1.getStudents();
for (Student student : students) {
  System.out.println("  Student's ID: " + student.getId() + ",
" + "Student's Name: " + student.getName());
}
Course course2 = session.get(Course.class, 2);
System.out.println("Course's ID: " + course2.getId());
System.out.println("Course's Name: " + course2.getName());
System.out.println("Course's Credit: " + course2.getCredit());
students = course2.getStudents();
for (Student student : students) {
  System.out.println("  Student's ID: " + student.getId() + ",
" + "Student's Name: " + student.getName());
}
```

Result


```
Course's ID: 1
Course's Name: Java
Course's Credit: 3
  Student's ID: 2, Student's Name: Somsri
  Student's ID: 1, Student's Name: Somchai
Course's ID: 2
Course's Name: Uml
Course's Credit: 3
  Student's ID: 2, Student's Name: Somsri
  Student's ID: 1, Student's Name: Somchai
```

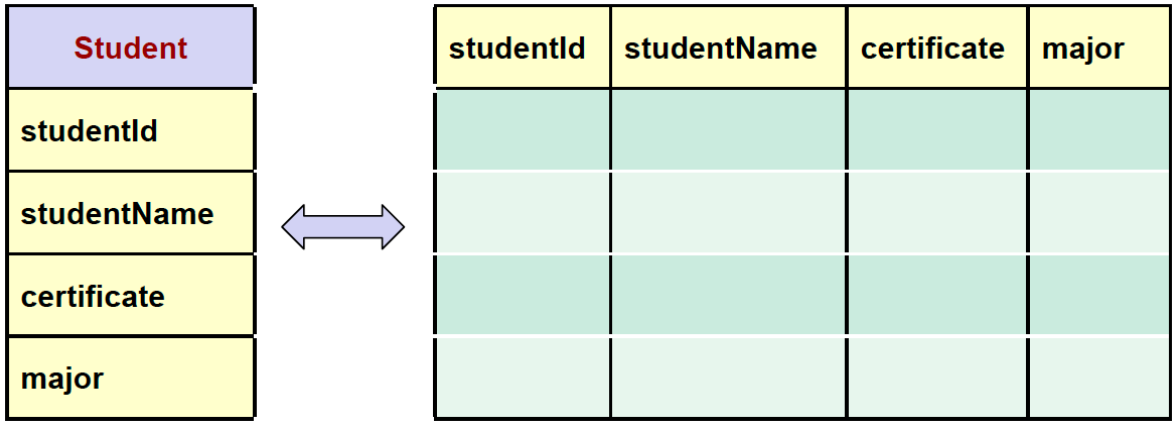
3. Embeddable Object

Entity Object vs Value Object

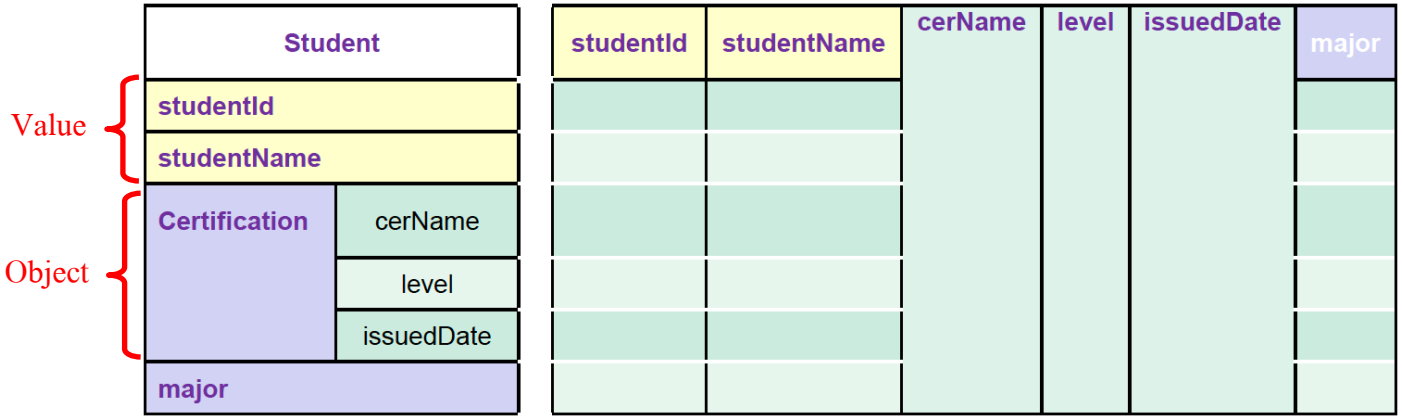
- Hibernate จัดแบ่งประเภทการใช้งานของอ็อบเจกต์ไว้ดังนี้
  - Entity อ็อบเจกต์มีช่วงชีวิตในการทำงานเป็นของตัวเองและสามารถนำไปใช้งานได้โดยตรง ในระดับของตาราง Entity อ็อบเจกต์ประกอบด้วยค่าที่ใช้เป็นคีย์หลัก และสามารถใช้อ้างอิงร่วมกับ Entity อ็อบเจกต์อื่นได้
  - Values อ็อบเจกต์ที่ไม่สามารถนำมาใช้งานได้โดยตรงแต่มักถูกใช้เป็นส่วนประกอบของ Entity อ็อบเจกต์อื่น ไม่มีช่วงชีวิตของตัวเองแต่มีช่วงชีวิตที่ขึ้นอยู่กับ Entity อ็อบเจกต์ที่ตัวเองเป็นส่วนประกอบ ในระดับตารางฐานข้อมูล Value อ็อบเจกต์จะมีค่าที่ไม่สามารถใช้เป็นคีย์หลักในตารางฐานข้อมูลได้

Entity Object

```
@Entity
@Table(name="students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int studentId;
  private String studentName;
  private String certificate;
  private String major;

  ...;
}
```



Entity & Value Object




```
@Entity
@Table (name="students")
public class Student {
    @Id @GeneratedValue
    @Column(name="student_id")
    private int studentId;
    @Column(name="student_name")
    private String studentName;
    private Certification certificate;
    @Column(name="student_major")
    private String major;
}
```

```
@Entity
public class Certification
{
    @Column(name="cert_name")
    private String cerName;
    @Column(name="level")
    private String level;
    @Column(name="issued_date")
    @Temporal(TemporalType.DATE)
    private Date issuedDate;
}
```

จะต้องระบุความสัมพันธ์ระหว่างคลาส

Embeddable Object

- เป็นการกำหนดตัวแปรอ็อบเจกต์ฝังตัวลงในคลาสที่ถูกนำไปใช้เป็นคอลัมน์ภายในตารางเอนติที โดยการเรียกใช้ @Embedded และ @Embeddable
- เช่น Entity คลาส (Student) ทำงานร่วมกับ Value คลาส (Cerification) ที่อยู่ในรูปของอ็อบเจกต์ฝังตัวมีโครงสร้างการทำงานบางส่วนดังต่อไปนี้

```
@Entity
public class Student {
   ....;
   @Embedded
   private Certification certificate;
}


@Embeddable
public class Certification ...{
     .....;
}
```

ในกรณีนี้เราทำให้เป็น Embedded Object

Student and Certiticate Classes


```
@Entity
@Table(name = "students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
  private int age;
  private String major;
  @Embedded
  private Certification certification;

  public Student(String name, int age, String major, Certification cer) {
    this.name = name;
    this.age = age;
    this.major = major;
    this.certification = cer;
  }

  ...;
}
```

```
@Embeddable
public class Certification {
  private String cerName;
  private String level;
  private Date issueDate;

  ...;
}
```

Run

```
SessionFactory sessionFactory = HibernateConnection.getSessionFactory();
Session session = sessionFactory.openSession();

try {
  session.beginTransaction();
  Certification cer = new Certification("Java Certified",
      "Intermediate", new Date());
  Student student = new Student("Somchai", 22,
                                "Information Technology", cer);

  session.save(student);
  session.getTransaction().commit();
} finally {
  if (session.getTransaction().isActive()) {
    session.getTransaction().rollback();
  }
  session.close();
}
```

Table: students

id	age	cerName	issueDate	level	major	name
1	22	Java Certified	2023-12-22 20:36:12	Intermediate	Information Technology	Somchai

Result

```
Student student = session.get(Student.class, 1);
System.out.println("Student's ID: " + student.getId());
System.out.println("Student's Name: " + student.getName());
System.out.println("Student's Age: " + student.getAge());
System.out.println("Student's Major: " + student.getMajor());
System.out.println("Student's Certification name: " +
                   student.getCertification().getCerName());
System.out.println("Student's Certification level: " +
                   student.getCertification().getLevel());
System.out.println("Student's Certification issue date: " +
                   student.getCertification().getIssueDate());
```

Student's ID: 1
Student's Name: Somchai
Student's Age: 22
Student's Major: Information Technology
Student's Certification name: Java Certified
Student's Certification level: Intermediate
Student's Certification issue date: 2023-12-22 21:12:59.0

4. Embeddable Object : Collections

Embeddable Object : Collections

- จำเป็นต้องเรียกใช้ @Embedded ร่วมกับ @ElementCollection และ @CollectionTable ดังรูปแบบการใช้งานต่อไปนี้

```
@Entity
@Table(name = "students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
  private int age;
  private String major;
  @ElementCollection
  @CollectionTable(name = "certifications",
                   joinColumns = @JoinColumn(name = "student_id"))
  private Collection<Certification> certificates =
                                        new ArrayList<Certification>();

  ...;
}
```

```
@Embeddable
public class Certification {
  private String cerName;
  private String level;
  private Date issueDate;

  ...;
}
```

Run

```
Certification cer1 = new Certification("Java Certified", "Intermediate",
                                       new Date());
Certification cer2 = new Certification("Oracle Certified", "Beginner",
                                       new Date());
Certification cer3 = new Certification("Test Certified", "Advanced",
                                       new Date());

Student student = new Student("Somchai", 22, "Information Technology");

student.getCertificates().add(cer1);
student.getCertificates().add(cer2);
student.getCertificates().add(cer3);

session.save(student);
```

Table: students

id	age	major	name
1	22	Information Technology	Somchai

Table: certifications

student_id	cerName	issueDate	level
1	Java Certified	2023-12-22 20:56:10	Intermediate
1	Oracle Certified	2023-12-22 20:56:11	Beginner
1	Test Certified	2023-12-22 20:56:12	Advanced

Result

```
Student student = session.get(Student.class, 1);

System.out.println("Student's ID: " + student.getId());
System.out.println("Student's Name: " + student.getName());
System.out.println("Student's Age: " + student.getAge());
System.out.println("Student's Major: " + student.getMajor());
Collection<Certification> certificates = student.getCertificates();

for (Certification cer : certificates) {
  System.out.println("Student's Certification name: " + cer.getCerName());
  System.out.println("Student's Certification level: " + cer.getLevel());
  System.out.println("Student's Certification issue date: " +
                     cer.getIssueDate());
}
```

Student's ID: 1
Student's Name: Somchai
Student's Age: 22
Student's Major: Information Technology
Student's Certification name: Java Certified
Student's Certification level: Intermediate
Student's Certification issue date: 2023-12-22 21:19:21.0
Student's Certification name: Oracle Certified
Student's Certification level: Beginner
Student's Certification issue date: 2023-12-22 21:19:21.0
Student's Certification name: Test Certified
Student's Certification level: Advanced
Student's Certification issue date: 2023-12-22 21:19:21.0

5. Composite Key

Composite Key

- ในกรณีที่ไม่มีการกำหนดคีย์หลักไว้ในตารางหรือบางครั้งการ
- สร้างคีย์หลักจากฟิลด์เดียวอาจมีโอกาสที่จะเกิดข้อมูลซ้ำกันได้ ดังนั้นจึงจำเป็นต้องใช้ฟิลด์อื่น ๆ ร่วมกันเพื่อใช้เป็นคีย์หลักที่มีค่าไม่ซ้ำกัน การใช้คีย์ร่วมกันในลักษณะดังกล่าวจะถูกเรียกว่าคีย์ผสม (Composite Key)
- Hibernate จึงได้ออกแบบกลไกสนับสนุนการสร้างคีย์ผสมซึ่งสามารถทำได้สองวิธี
  - วิธีแรกใช้สัญลักษณ์ @IdClass
  - วิธีที่สองใช้สัญลักษณ์ @EmbeddedId

Composite Key: Considerations

- คลาสที่เป็นคีย์หลักจะต้องประกาศเป็นแบบ public หรือ protected และเป็นคอนสตรัคเตอร์แบบ Default เพื่อให้สามารถเข้าถึงได้
- คลาสที่เป็นคีย์หลักต้องเป็นแบบ Serializable
- คลาสที่เป็นคีย์หลัก ต้องประกาศเมธอด equals() และ hashCode() ไว้เสมอ (ในกรณีที่ใช้ร่วมกับ Collections)
- คีย์ผสมต้องถูกประกาศให้อยู่ในรูปของ embeddable คลาสหรือถูกแปลงให้เป็นแอททริบิวต์ต่าง ๆ ของ Entity คลาส
- ในกรณีที่คีย์ผสมถูกแปลงให้เป็นแอททริบิวต์ต่าง ๆ ของ Entityคลาส ชื่อของคีย์หลักของฟิลด์ภายในคลาสที่เป็นคีย์หลักและ Entity คลาสเหล่านั้นจะต้องเกี่ยวข้องและมีชนิดข้อมูลที่เหมือนกัน

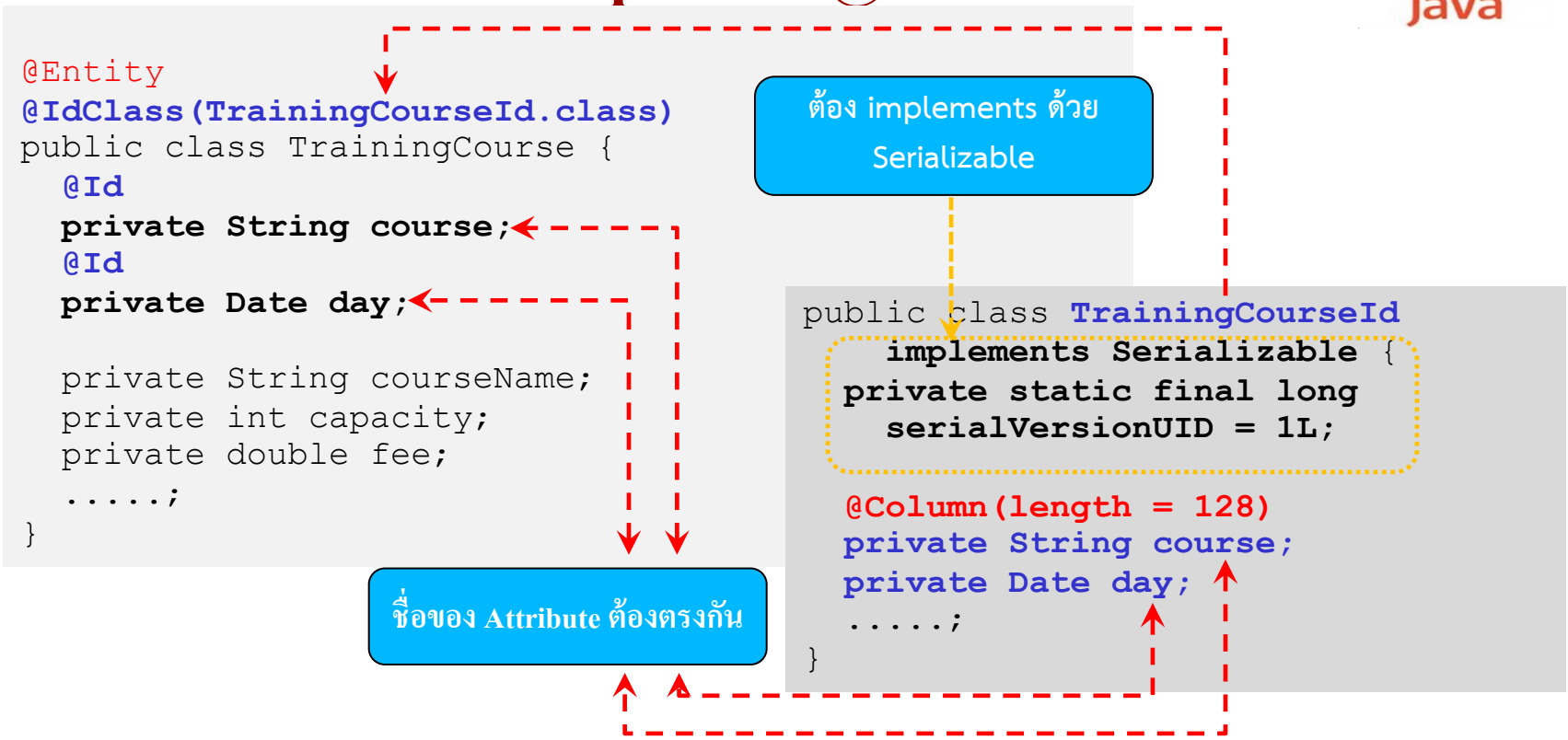
Composite : @IdClass

- สัญลักษณ์ @IdClass ใช้ในการสร้างคีย์ผสมภายใน Entity คลาส โดยอ้างอิงจากแอททริบิวต์ที่ถูกกำหนดอยู่ภายใน Embeddable คลาส
- ชื่อแอททริบิวต์ที่ถูกระบุภายใน Entity คลาสต้องมีชื่อและชนิดข้อมูลเหมือนกับแอททริบิวต์ภายใน Embeddable คลาสเสมอ เช่น คลาส TrainningCourse ที่ใช้คลาส Schedule เป็นคีย์ผสม

TrainingCourse	
Schedule	couse
	day
courseName	
capacity	
fee	

course	day	courseName	capacity	fee

Composite : @IdClass



- @IdClass(Schedule.class) ถูกกำหนดไว้ใน Entity คลาส (TrainingCourse) ที่มีการอ้างอิงไปยังคลาส Schedule
- Schedule ถูกกำหนดให้อยู่ในรูปของ Embeddable คลาส
  - ประกาศตัวแปร course และ day พร้อมสัญลักษณ์ @Id เพื่อใช้สำหรับการนำเสนอคีย์ผสมไว้ภายในคลาส

```
public class TrainingCourseId implements Serializable {
  private static final long serialVersionUID = 1L;

  private String course;
  private Date day;

  public TrainingCourseId() {}

  public TrainingCourseId(String course, Date day) {
    this.course = course;
    this.day = day;
  }

  @Override
  public int hashCode() {
    ...
  }

  @Override
  public boolean equals(Object obj) {
    ...
  }

}
```

เนื่องจากเราใช้ TrainingCourseId เป็น ClassId จำเป็นต้อง override method สองตัวได้แก่ hashCode() และ equals()

เราสามารถใช้ IDE ช่วยสร้าง hashCode() และ equals() ได้ เช่นใน Visual Studio Code ให้คลิกขวาเลือก Source Action... -> Generate hashCode and equals()... และเลือก attributes ที่ต้องการ


```
public class DateUtils {
  private static DateFormat dateFormat =
                              new SimpleDateFormat("dd-MM-yyyy");

  public static Date StringToDate(String dateString) {
    Date date = null;
    try {
      // Parse the string to a Date object
      date = dateFormat.parse(dateString);
    } catch (ParseException e) {
      e.printStackTrace();
    }
    return date;
  }
}
```

คลาสสำหรับแปลงวันที่จาก String ให้เป็น Date

"20-04-2023" ==> "2023-04-20 00:00:00"

String

Date Object

```
TrainingCourse course1 = new TrainingCourse("DB 101",
DateUtils.StringToDate("20-4-2023"), "Relational Database",
    30,
    1500);
TrainingCourse course2 = new TrainingCourse("Java 102",
DateUtils.StringToDate("24-4-2023"), "OO Programming", 40,
    2500);

session.save(course1);
session.save(course2);
```

Run Code

```
TrainingCourseId id1 = new TrainingCourseId("DB 101",
DateUtils.StringToDate("20-4-2023"));
TrainingCourse course = session.get(TrainingCourse.class, id1);
System.out.println(course.getCourseName());
System.out.println(course.getCapacity());
System.out.println(course.getFee());

TrainingCourseId id2 = new TrainingCourseId("Java 102",
DateUtils.StringToDate("24-4-2023"));
TrainingCourse course2 = session.get(TrainingCourse.class, id2);
System.out.println(course2.getCourseName());
System.out.println(course2.getCapacity());
System.out.println(course2.getFee());
```

```
Hibernate: insert into TrainingCourse (capacity, courseName, fee, course, day) values (?, ?, ?, ?, ?)
Hibernate: insert into TrainingCourse (capacity, courseName, fee, course, day) values (?, ?, ?, ?, ?)
```

ผลลัพธ์:

```
Relational Database
30
1500.0
OO Programming
40
2500.0
```

Table: training_courses

course	day	capacity	courseName	fee
DB 101	2023-04-20 00:00:00	30	Relational Database	1500
Java 102	2023-04-24 00:00:00	40	OO Programming	2500

Composite : @IdClass

- การเรียกใช้สัญลักษณ์ @IdClass ไว้ภายใน Entity คลาสมีสิ่งที่น่าสนใจที่สามารถสรุปได้ดังต่อไปนี้
  - คีย์ผสมได้แก่ แอททริบิวต์ course และ day ภายในคลาสคีย์หลักจะต้องปรากฏอยู่ใน Entity คลาส (TrainingCourse) ที่มีการระบุสัญลักษณ์ @Id ไว้
  - คลาส Schedule ต้องไม่มีการกำหนดเมธอดประเภท setter ไว้เนื่องจากคีย์ผสมไม่มีการเปลี่ยนแปลงค่าในภายหลัง
  - ในบางกรณีอาจมีการโอเวอร์ไรด์เมธอด equals() และ hashcode() เนื่องจากอ็อบเจกต์ที่เป็นคีย์หลักต้องจำแนกความแตกต่างออกจากอ็อบเจกต์อื่น ๆ ได้ เพื่อให้สามารถนำไปใช้ได้กับ Collections ที่มีการใช้คุณสมบัติ hashing ได้

6. Composite : @EmbeddedId

Composite : @EmbeddedId

- ใช้สัญลักษณ์ @EmbeddedId ร่วมกับ Embeddedable คลาสเพื่อสร้างอ็อบเจกต์ฝังตัวที่ถูกนำมาใช้งานในรูปของคีย์ผสม

```
@Entity
public class TrainingCourse {
    @EmbeddedId
    private TrainingCourseId schedule;
    .....;
}
```

```
@Embeddable
public class TrainingCourseId implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    .......
    .......
}
```

Example


```
TrainingCourseId id1 = new TrainingCourseId("DB 101",
                                DateUtils.StringToDate("20-4-2023"));
TrainingCourse course1 = new TrainingCourse(id1, "Relational Database",
    30,
    1500);

TrainingCourseId id2 = new TrainingCourseId("Java 102",
                                DateUtils.StringToDate("24-4-2023"));
TrainingCourse course2 = new TrainingCourse(id2, "OO Programming",
    40,
    2500);

session.save(course1);
session.save(course2);
```

Run Code

```
TrainingCourseId id1 = new TrainingCourseId("DB 101",
DateUtils.StringToDate("20-4-2023"));
TrainingCourse course = session.get(TrainingCourse.class, id1);
System.out.println(course.getCourseName());
System.out.println(course.getCapacity());
System.out.println(course.getFee());

TrainingCourseId id2 = new TrainingCourseId("Java 102",
DateUtils.StringToDate("24-4-2023"));
TrainingCourse course2 = session.get(TrainingCourse.class, id2);
System.out.println(course2.getCourseName());
System.out.println(course2.getCapacity());
System.out.println(course2.getFee());
```

```
Hibernate: insert into TrainingCourse (capacity, courseName, fee, course, day) values (?, ?, ?, ?, ?)
Hibernate: insert into TrainingCourse (capacity, courseName, fee, course, day) values (?, ?, ?, ?, ?)
```

ผลลัพธ์:

```
Relational Database
30
1500.0
OO Programming
40
2500.0
```

Table: training_courses

course	day	capacity	courseName	fee
DB 101	2023-04-20 00:00:00	30	Relational Database	1500
Java 102	2023-04-24 00:00:00	40	OO Programming	2500

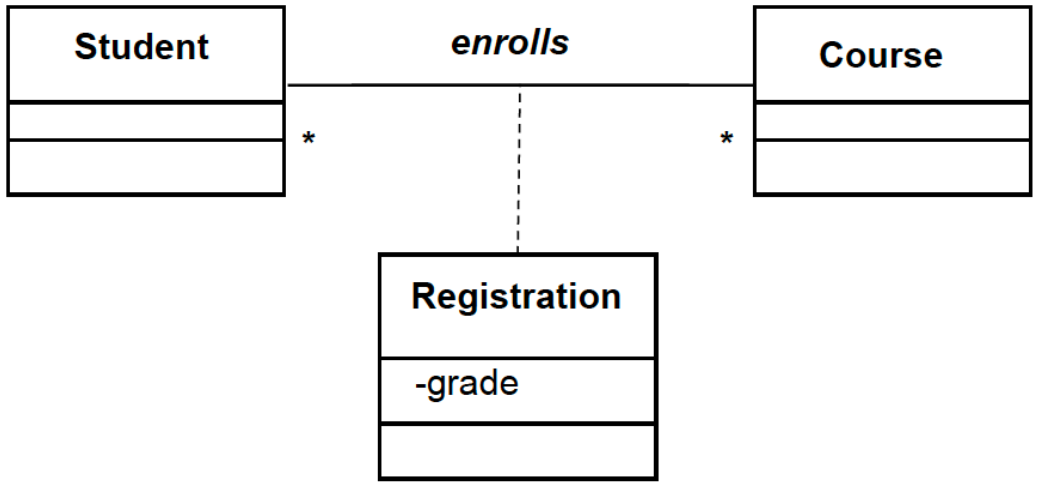
7. Extra Column : Many To Many

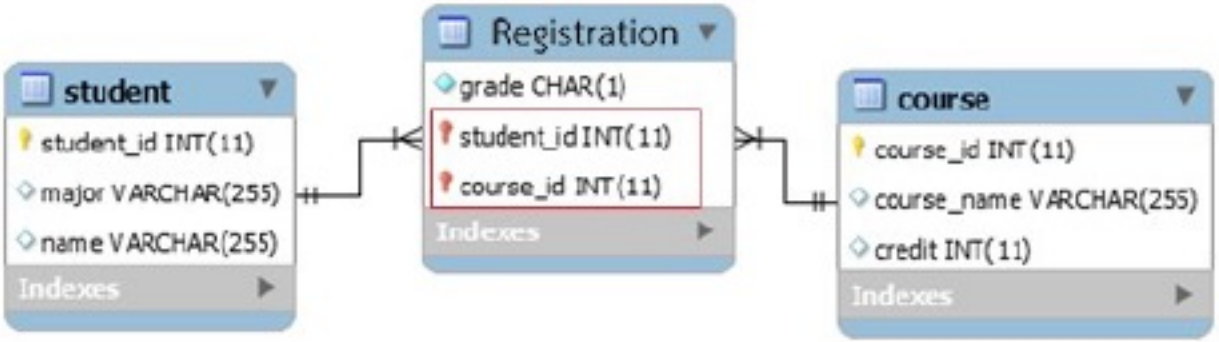
Association Class

- ความสัมพันธ์แบบกลุ่มต่อกลุ่มอาจมีปัญหาเกิดขึ้นในกรณีที่ต้องการเพิ่มแอททริบิวต์เพื่อแสดงผลการเรียน (grade) ซึ่งอาจมีการทำงานได้เป็นสองกรณี
  - กรณีแรกกำหนดแอททริบิวต์ grade ลงในคลาส Student
  - กรณีที่สองกำหนด grade ลงในคลาส Course
  - ปัญหาที่อาจจะเกิดขึ้นคือ
    - หากนักศึกษาลงทะเบียนเรียนหลายวิชาผลเกรดจะเก็บไว้ที่ไหน ?
      - เก็บไว้ที่ Student ปัญหาที่เกิดขึ้นคือ ?
      - เก็บไว้ที่ Course ปัญหาที่เกิดขึ้นคือ ?

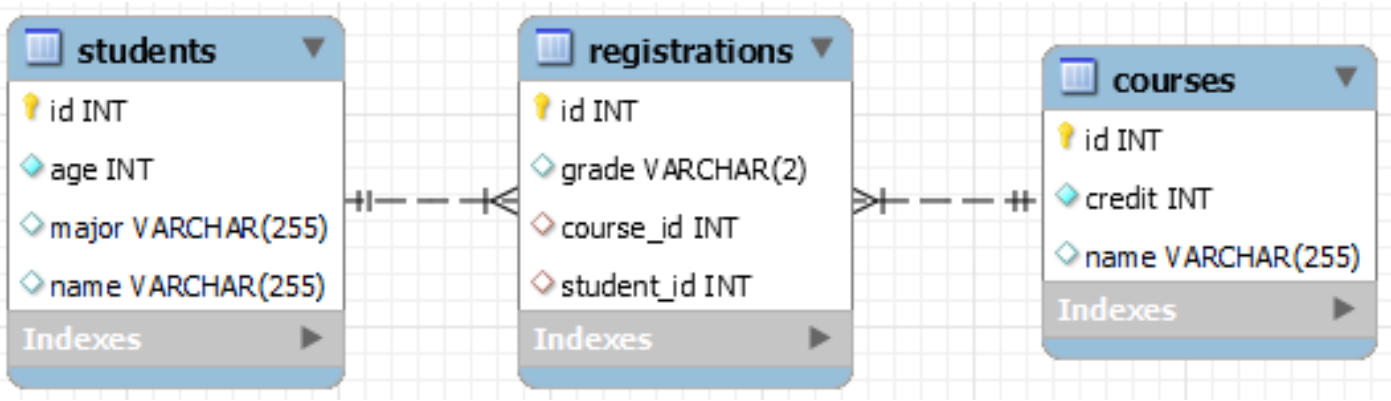
Association Class

- ดังนั้นจึงแก้ไขความสัมพันธ์โดยใช้ AssociationClass ดังต่อไปนี้





Extra Column : Many To Many



- ความสัมพันธ์แบบกลุ่มต่อกลุ่มจะถูกสร้างขึ้นผ่านตารางที่สามที่ประกอบไปด้วย Foreign Key ที่ใช้อ้างอิงกลับไปยัง FK ของสองตารางแรก
- ในกรณีที่มีการเพิ่มฟิลด์ grade
  - สามารถเพิ่มลงในตารางที่สามได้ รวมถึงฟิลด์อื่นที่เกี่ยวข้องกับผลการเรียนอื่น ๆ

Extra Column : Many To Many

- คลาส Student และ Course มีลักษณะเดียวกับตัวอย่างการทำงานในความสัมพันธ์แบบกลุ่มต่อกลุ่ม
- สร้างคลาส StudentCourseId ทำหน้าที่เป็นคีย์หลักในตารางที่สาม
  - เพื่อใช้เป็นคีย์ผสม และต้องระบุความสัมพันธ์แบบกลุ่มต่อหนึ่งที่มีการอ้างอิงไปยัง Student และ Course อ็อบเจกต์
  - กำหนดสัญลักษณ์ @Embeddable ไว้เพื่อให้ผู้ใช้สามารถนำคลาสดังกล่าวไปใช้ในการอ้างอิงแบบฝังตัวกับคลาส Registration

Extra Column : Many To Many

```
@Entity
public class Student {
    .....
    @OneToMany(mappedBy = "id.student", cascade = CascadeType.ALL)
    private Set<Registration> registrations = new HashSet<Registration>();
```

```
@Entity
public class Course {
    .....
    private int courseId;
    @OneToMany(mappedBy = "id.course")
    private Set<Registration> registrations = new HashSet<Registration>();
```

อ้างถึงตัวแปร course ที่อยู่ในคลาสฝั่งตัว

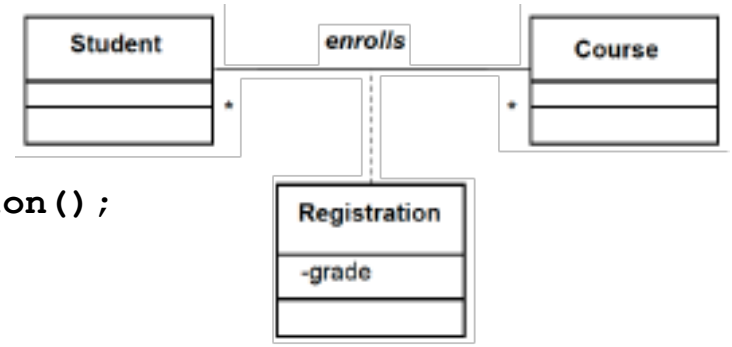
Extra Column : Many To Many


```
@Entity
public class Registration {
   @EmbeddedId
   private RegistrationId id = new RegistrationId();
   private char grade;
   public StudentCourseId getId() {
      return id;
   }
```

```
@Embeddable
public class RegistrationId implements Serializable {
   private static final long serialVersionUID = 1L;
   @ManyToOne
   private Student student;
   @ManyToOne
   private Course course;
}
```

Run

```
Student student = new Student("Somsri Jaipak", 22, "IT");
Course course1 = new Course("Java", 3);
Course course2 = new Course("Uml", 3);
session.save(course1);
session.save(course2);
session.save(student);

Registration registration1 = new Registration();
registration1.getId().setStudent(student);
registration1.getId().setCourse(course1);
registration1.setGrade("B+");
Registration registration2 = new Registration();
registration2.getId().setStudent(student);
registration2.getId().setCourse(course2);
registration2.setGrade("B");

// Student register courses
student.getRegistrations().add(registration1);
student.getRegistrations().add(registration2);

session.save(student);
```



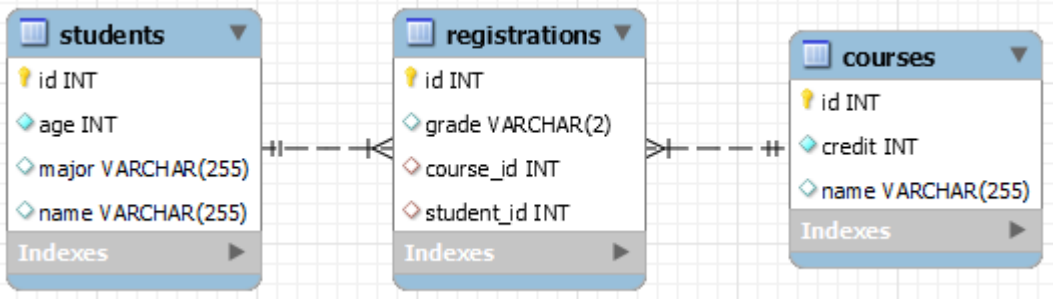


Table: courses

id	credit	name
1	3	Java
2	3	Uml

Table: students

id	age	major	name
1	22	IT	Somsri Jaipak

Table: registrations

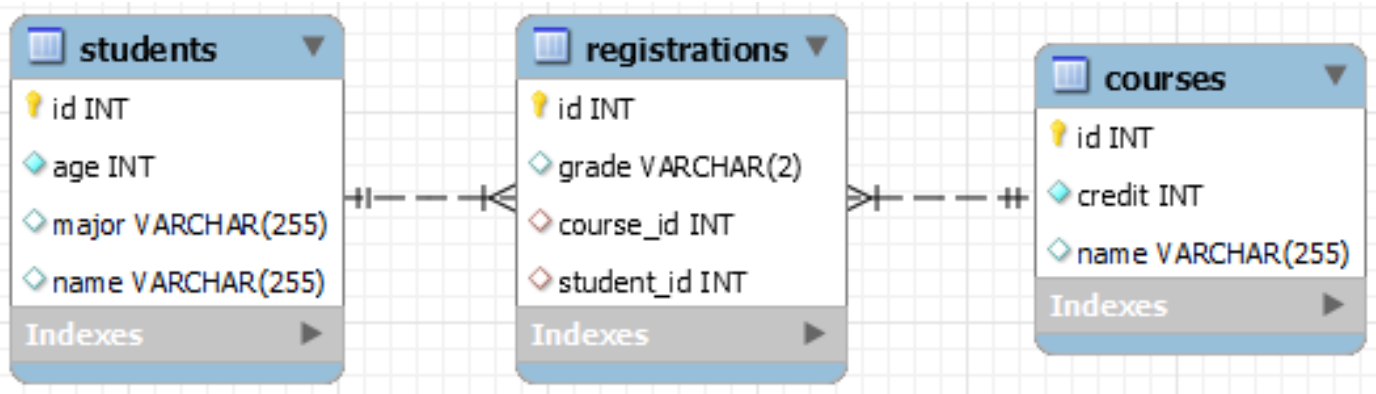
grade	student_...	course_id
B+	1	1
B	1	2

```
System.out.println("=====================================");
Student student = (Student) session.get(Student.class, 1);
System.out.println("Student name: " + student.getName());
System.out.println("Student age: " + student.getAge());
System.out.println("Student major: " + student.getMajor());
System.out.println("Student registered courses: ");
for (Registration registration : student.getRegistrations()) {
  System.out.println("Course name: " +
                     registration.getId().getCourse().getName());
  System.out.println("Course credit: " +
                     registration.getId().getCourse().getCredit());
  System.out.println("Grade: " + registration.getGrade());
}
System.out.println("=====================================");
Course course = (Course) session.get(Course.class, 1);
System.out.println("Course name: " + course.getName());
System.out.println("Course credit: " + course.getCredit());
System.out.println("Course registered students: ");
for (Registration registration : course.getRegistrations()) {
  System.out.println("Student name: " +
                     registration.getId().getStudent().getName());
  System.out.println("Student age: " +
                     registration.getId().getStudent().getAge());
  System.out.println("Student major: " +
                     registration.getId().getStudent().getMajor());
  System.out.println("Grade: " + registration.getGrade());
}
System.out.println("=====================================");
```

8. Extra Column M To M : Without Composite

Extra Column M To M : Without Composite

- ในกรณีที่ไม่ใช้ Composite Key
  - จำเป็นต้องกำหนดคีย์หลักในคลาส Registration
  - ที่คลาส Registration ให้กำหนด @ManyToOne ไปยังคลาส Student และ Course
  - ที่คลาส Student และ Course ให้กำหนด @OneToMany ไปยังคลาส Registration เพื่อให้เป็น Bidirectional Relationship



Extra Column M To M : Without Composite


```
@Entity
public class Student {
    .....
    @OneToMany(mappedBy = "student")
    private Set<Registration> registrations = new HashSet<Registration>();
```

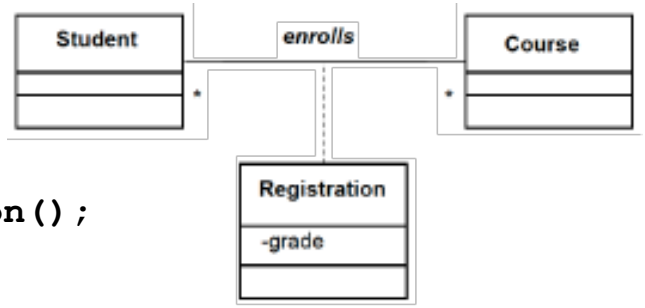
```
@Entity
public class Course {
    .....
    @OneToMany(mappedBy = "course")
    private Set<Registration> registrations = new HashSet<Registration>();
```

Extra Column M To M : Without Composite


```
@Entity
public class Registration {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "STUDENT_ID")
    private Student student;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "COURSE_ID")
    private Course course;
}
```

Run

```
Student student = new Student("Somsri Jaipak", 22, "IT");
Course course1 = new Course("Java", 3);
Course course2 = new Course("Uml", 3);
session.save(course1);
session.save(course2);
session.save(student);

Registration registration1 = new Registration();
registration1.setStudent(student);
registration1.setCourse(course1);
registration1.setGrade("B+");
Registration registration2 = new Registration();
registration2.setStudent(student);
registration2.setCourse(course2);
registration2.setGrade("B");

// Student register courses
student.getRegistrations().add(registration1);
student.getRegistrations().add(registration2);

session.save(student);
```



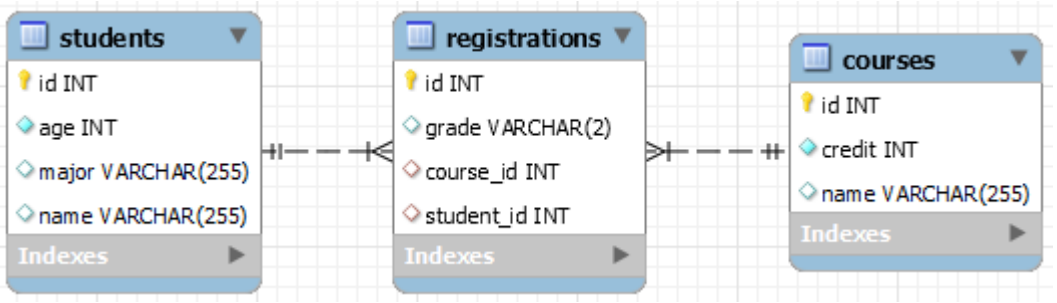


Table: courses

id	credit	name
1	3	Java
2	3	Uml

Table: students

id	age	major	name
1	22	IT	Somsri Jaipak

Table: registrations

id	grade	course_id	student_id
1	B	2	1
2	B+	1	1


```
System.out.println("====================================");
Student student = (Student) session.get(Student.class, 1);
System.out.println("Student name: " + student.getName());
System.out.println("Student age: " + student.getAge());
System.out.println("Student major: " + student.getMajor());
System.out.println("Student registered courses: ");
for (Registration registration : student.getRegistrations()) {
  System.out.println("Course name: " +
                     registration.getCourse().getName());
  System.out.println("Course credit: " +
                     registration.getCourse().getCredit());
  System.out.println("Grade: " + registration.getGrade());
}
System.out.println("====================================");
Course course = (Course) session.get(Course.class, 1);
System.out.println("Course name: " + course.getName());
System.out.println("Course credit: " + course.getCredit());
System.out.println("Course registered students: ");
for (Registration registration : course.getRegistrations()) {
  System.out.println("Student name: " +
                     registration.getStudent().getName());
  System.out.println("Student age: " +
                     registration.getStudent().getAge());
  System.out.println("Student major: " +
                     registration.getStudent().getMajor());
  System.out.println("Grade: " + registration.getGrade());
}
System.out.println("====================================");
```

Good luck

Refference:

https://hibernate.org/orm/releases/5.6/

https://hibernate.org/orm/documentation/5.6/

https://docs.jboss.org/hibernate/stable/orm/userguide/html_single/Hibernate_User_Guide.html