www.bilgem.tubitak.gov.tr
2023

TÜBİTAK BİLİŞİM VE BİLGİ GÜVENLİĞİ İLERİ TEKNOLOJİLER ARAŞTIRMA MERKEZİ

HIBERNATE – JPA - ORM

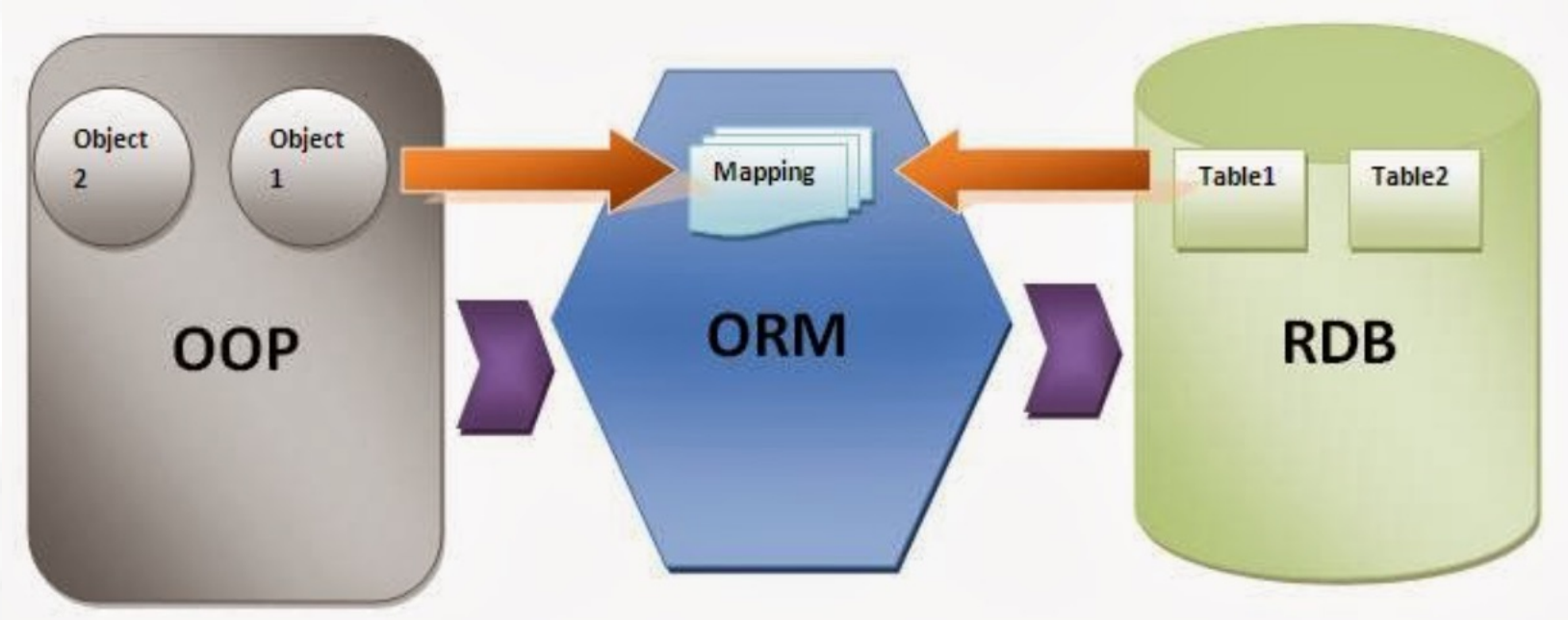
Persistence Nedir?

- Persistence kavramı, nesnelerin veritabanında kalıcı bir şekilde saklanabilmesini ifade eder.

- Bir nesne persistent olduğunda, veritabanında depolanmasına(store) ve geril yüklenmesine(load) izin verilir.

JPA ve Hibernate Nedir?

- Özetle, nesne/ilişkisel eşleme, bir Java uygulamasındaki nesnelerin, uygulamanın sınıfları ile SQL veritabanının şeması arasındaki eşlemeyi tanımlayan meta verileri kullanarak bir SQL veritabanındaki tablolara otomatik işlenmesini sağlayan bir araçtır.



ORM(Object Relational Mapping)

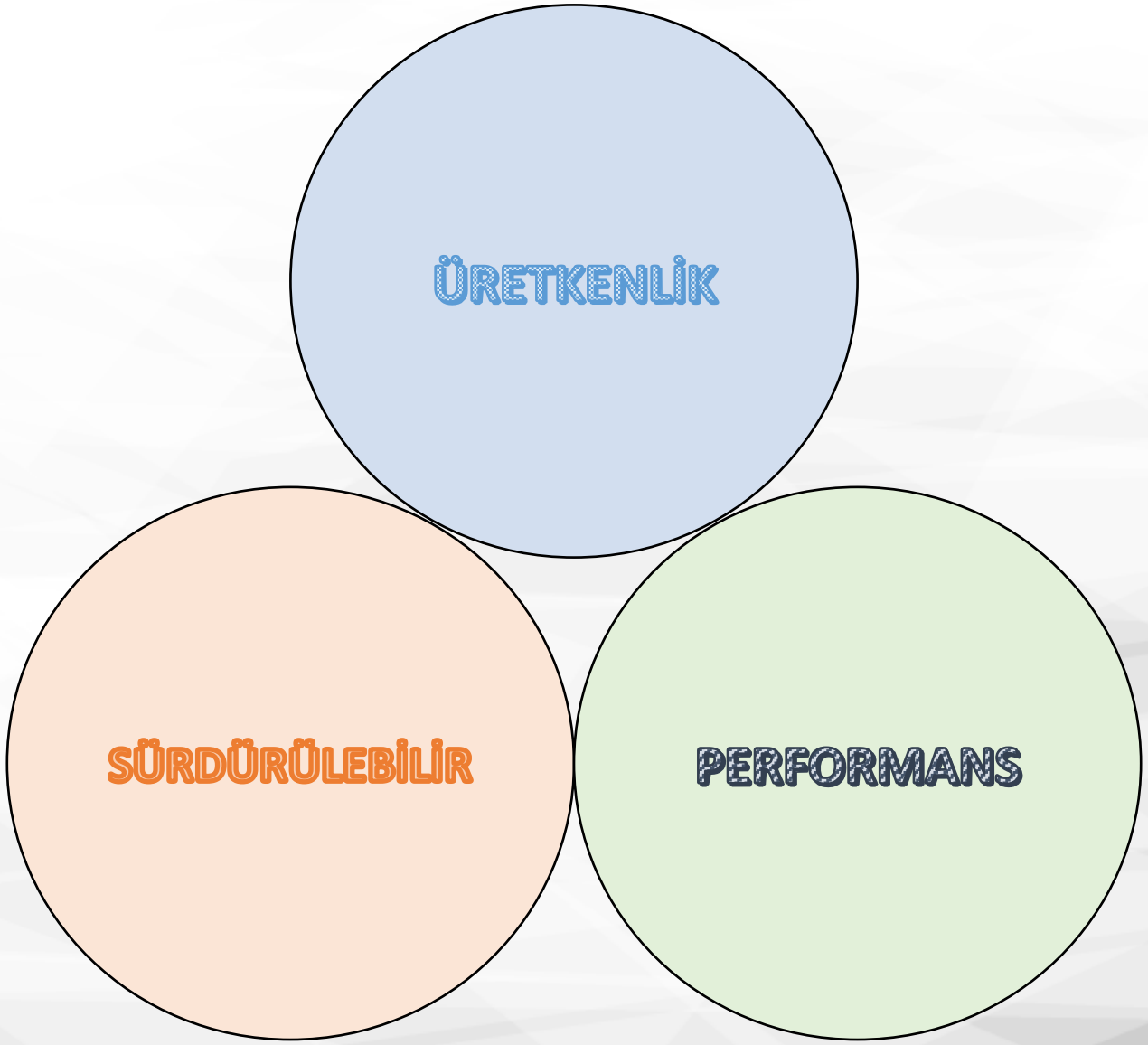
JDBC


```
public class PersistenceTest {
    Connection getConnection() throws SQLException {
        return DriverManager.getConnection("jdbc:h2:tcp://localhost/~/test", "sa","");
    }

    @BeforeClass
    public void setup() {
        final String DROP = "DROP TABLE users IF EXISTS";
        final String CREATE = "CREATE TABLE users ("
                + "id BIGINT GENERATED BY DEFAULT AS IDENTITY "
                + "PRIMARY KEY, "
                + "name VARCHAR(256) NOT NULL)";
        try (Connection connection = getConnection()) {
            // clear out the old data, if any, so we know the state of the DB
            try (PreparedStatement ps =
                        connection.prepareStatement(DROP)) {
                ps.execute();
            }
            // create the table...
            try (PreparedStatement ps =
                        connection.prepareStatement(CREATE)) {
                ps.execute();
            }
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
```

```java
public UserEntity saveUser(String name) {
    final String INSERT = "INSERT INTO users(name) VALUES (?)";
    UserEntity message = null;
    try (Connection connection = getConnection()) {
        try (PreparedStatement ps =
                     connection.prepareStatement(INSERT,
                             Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, name);
            ps.execute();
            try (ResultSet keys = ps.getGeneratedKeys()) {
                if (!keys.next()) {
                    throw new SQLException("No generated keys");
                }
                message = new UserEntity(keys.getLong(1), name);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
    return message;
}
```

```java
@Test
public void readUser() {
    final String name = "Baran SÖNMEZ";
    UserEntity userEntity = saveUser(name);
    final String SELECT = "SELECT id, name FROM users";
    List<UserEntity> list = new ArrayList<>();
    try (Connection connection = getConnection()) {
        try (PreparedStatement ps =
                     connection.prepareStatement(SELECT)) {
            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    UserEntity newUser = new UserEntity();
                    newUser.setId(rs.getLong(1));
                    newUser.setName(rs.getString(2));
                    list.add(userEntity);
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
    assertEquals(list.size(), 1);
    for (UserEntity m : list) {
        System.out.println(m);
    }
    assertEquals(list.get(0), userEntity);
}
```

Hibernate ve JPA Bize Ne Sağlar?



NESNE MODEL - ENTITY MODEL UYUMSUZLUĞU

Nesne Model – İlişkisel Model Uyuşmazlığı

Java Kodu

```java
public class Category {

    private Long categoryId;
    private String categoryName;
    private Set<Product> products;
}

public class Product {

    private Long productId;
    private String productName;
    private double price;
    private Category category;
}
```

Veritabanı Yapısı

```sql
CREATE TABLE Categories (
  category_id BIGINT PRIMARY KEY,
  category_name VARCHAR(100) NOT NULL
);


CREATE TABLE Products (
  product_id BIGINT PRIMARY KEY,
  product_name VARCHAR(200) NOT NULL,
  price DECIMAL NOT NULL,
  FOREIGN KEY (category_id) REFERENCES Categories(category_id)
);
```

Nesne - Tablo Yapısı

Java Kodu

```java
public class User {

    private String username;
    private Address address;
}


public class Address {

    private String city;
    private String street;
    private String zip;
}
```

Veritabanı Yapısı

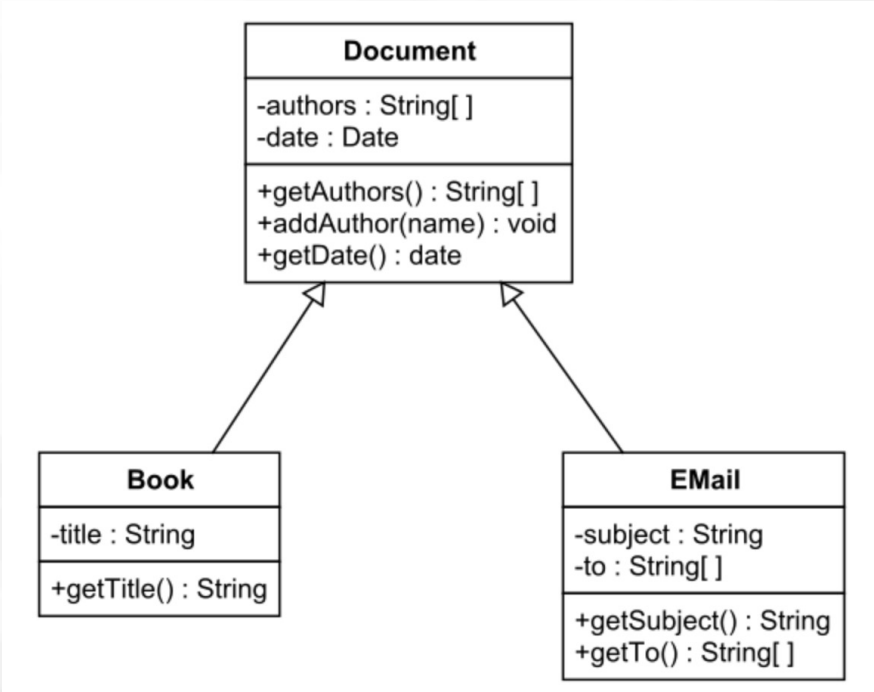
```
create table USERS (
    USERNAME varchar(15) not null primary key,
    ADDRESS_STREET varchar(255) not null,
    ADDRESS_ZIPCODE varchar(5) not null,
    ADDRESS_CITY varchar(255) not null
);
```

**Nesne modeldeki her sınıf ilişkisel modelde bir tabloya karşılık gelmeyebilir.

Inheritance problemi

OOP'için en önemli konulardan biri de kalıtım diyebiliriz. Java uygulamalarında kalıtımı extends keyword'ü kullanarak kolaylıkla uygulayabilirz.

Nesne modelde hiyerarşi söz konusudur, ilişkisel modelde tablolar arasında hiyerarşi yoktur.



Identity Problemi

Nesne modelde iki tür eşitlik vardır.

1- ==

2- Object.equals()

İlişkisel modelde primary key bilgisi var

İlişkilerin Yönetilmesi

Nesne modelde ilişkilerin yön bilgisi vardır.

person -> adres veya adres -> person (tek veya çift yönlü olabilir.)

İlişkisel modelde yönlü bir ilişki yoktur.

```
public class Student {

    private List<Course> courses;
}

public class Course {

    private List<Student> students;
}
```

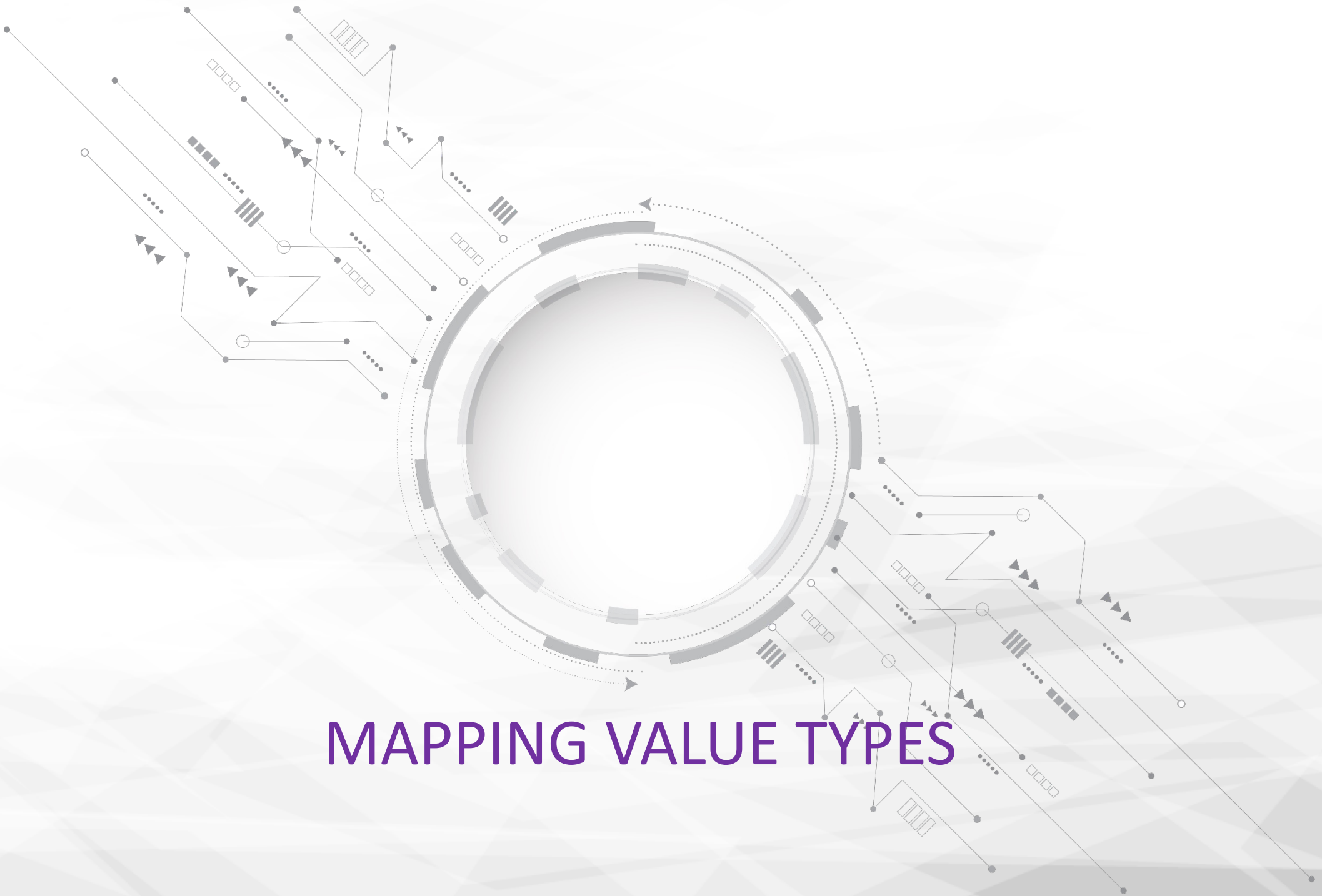
```
CREATE TABLE Student_Course (
  student_id BIGINT,
  course_id BIGINT,
  PRIMARY KEY (student_id, course_id)
  FOREIGN KEY (student_id) REFERENCES Students(student_id),
  FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Veriye Ulaşmak

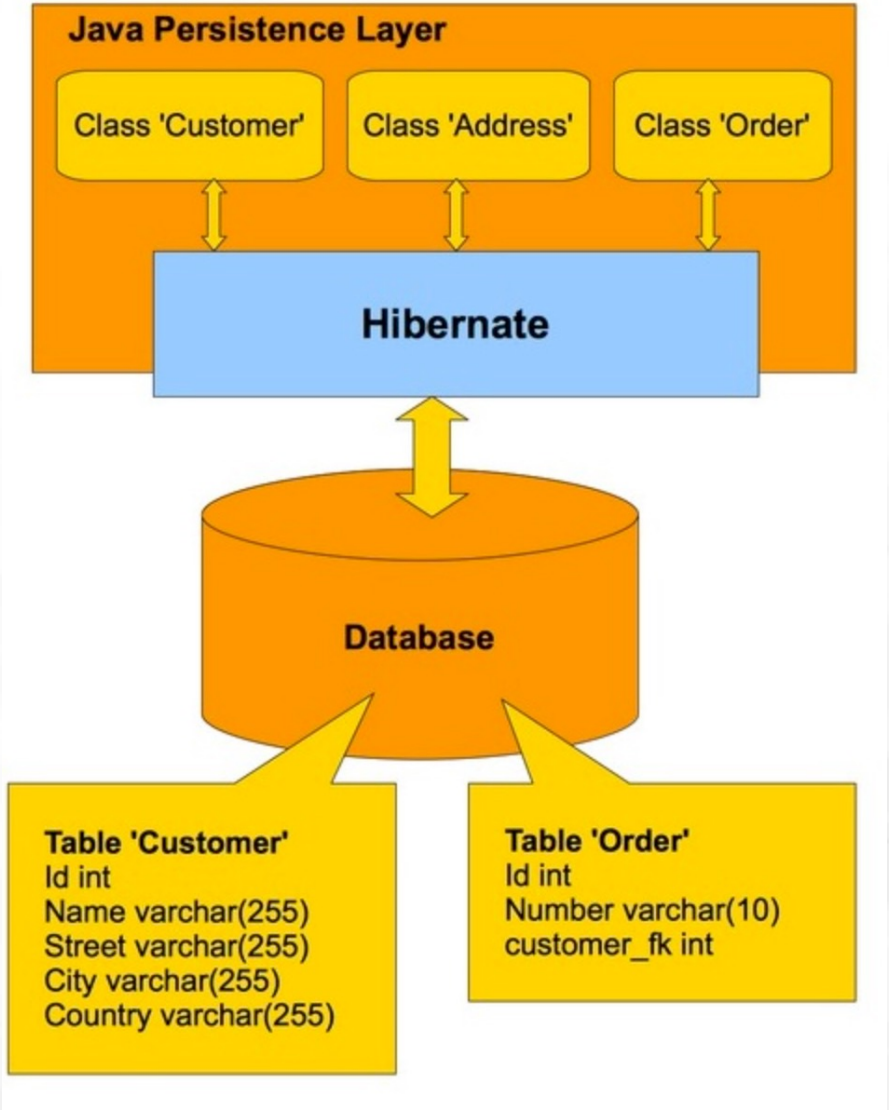
- Java'da veriye ulaşmak :
  - studentNesnesi.getCourses();

- Veritabanında join kullanmamız gerekebilir.

```
SELECT * FROM Students s
    LEFT OUTER JOIN Courses c
    ON c.student_id = s.id
WHERE s.id = 1;
```



MAPPING VALUE TYPES


Java Persistence Layer
Class 'Customer'
Class 'Address'
Class 'Order'
Hibernate
Database
Table 'Customer'
Id int
Name varchar(255)
Street varchar(255)
City varchar(255)
Country varchar(255)
Table 'Order'
Id int
Number varchar(10)
customer_fk int

Hibernate Entity Data Type	SQL Data Type
Integer	INT or INTEGER
Long	BIGINT
Double	DOUBLE or FLOAT
BigDecimal	DECIMAL or NUMERIC
String	VARCHAR or TEXT
Character	CHAR or VARCHAR
Date	DATETIME or TIMESTAMP
LocalDate	DATE
LocalTime	TIME
Boolean	BIT or BOOLEAN
Enum	DECIMAL or NUMERIC
Byte or Byte[]	BYTEA or BLOB

Mapping types

Annotations

- @Entity
- @Table
- @Id
- @GeneratedValue
- @Column

```java
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import jakarta.persistence.GeneratedValue;

no usages   new *
@Entity
@Table(name = "books")
public class Book {

    @Id
    @GeneratedValue
    private Long id;

    no usages
    @Column(name = "title")
    private String title;
}
```

Hibernate.configuration.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!--  Database connection settings  -->
        <property name="hibernate.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:~/test</property>
        <property name="hibernate.connection.username">your_username</property>
        <property name="hibernate.connection.password">your_password</property>

        <!-- Hibernate dialect for H2 -->
        <property name="dialect">org.hibernate.dialect.H2Dialect</property>

        <property name="hbm2ddl.auto">create-drop</property>

        <!-- Mapping files -->
        <!-- Add your entity class mappings here -->
        <mapping class="pojo.StudentEntity"/>

    </session-factory>
</hibernate-configuration>
```

Session management

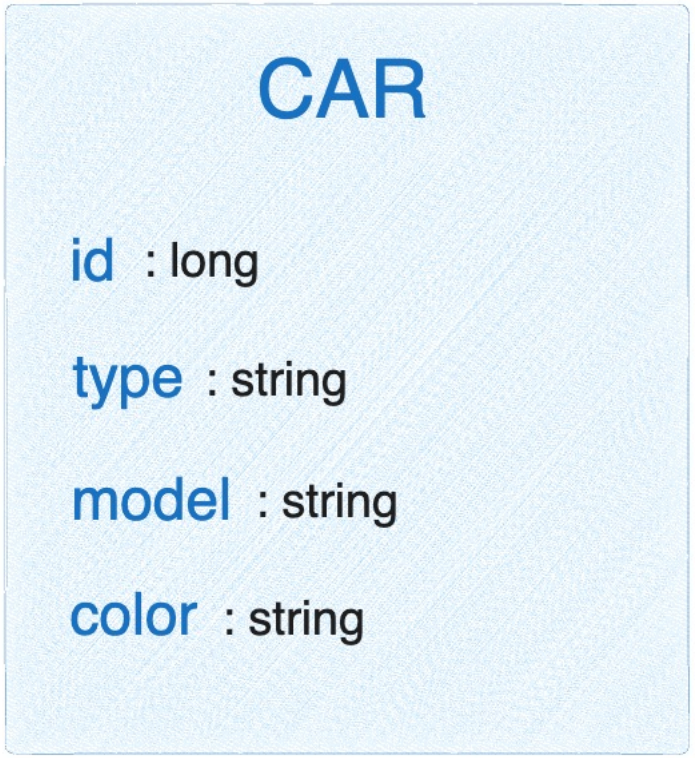
```
private SessionFactory factory = null;

new *
@BeforeClass
public void setup() {
    StandardServiceRegistry registry =
            new StandardServiceRegistryBuilder()
                    .configure()
                    .build();
    factory = new MetadataSources(registry)
            .buildMetadata()
            .buildSessionFactory();


    try (Session session = factory.openSession()) {
        Transaction tx = session.beginTransaction();
        //session.persist();
        tx.commit();
    }
}
```

- Session Factory
- Session
- Transaction
  - commit()
  - rollback()

KURULUM

Repo: https://github.com/ezgiturk/hibernate-starter

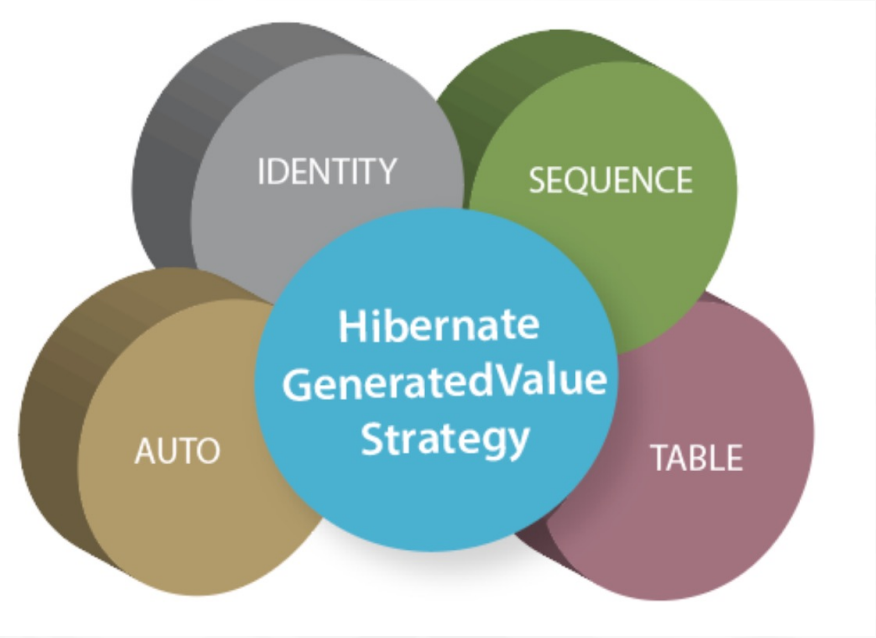
ÖRNEK - 1



- ✓ Test metodumuzda bu entityden bir nesne oluşturalım .
- ✓ persist() metodu ile veritabanına kaydedelim.
- ✓ find() metodu ile veritabanından getirip console'a yazdıralım.
- ✓ hibernate configuration ayarını unutmayın!

Identity

Primary key üretimi için kullanılır.



Enumerated Type (String, Ordinal)

- Enumerated EnumType ile beraber kullanılır.
- EnumType string ise enum'un string değeri veritabanına yazılır.
- Ordinal ise sayısal bir değer yazılır.

✓ Sınırlı Değerler

✓ Sabitler

✓ Durumlar

```java
@Getter
@NoArgsConstructor
@ToString
@Entity
public class Article {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Enumerated(EnumType.ORDINAL)
    private Status status;

    @Enumerated(EnumType.STRING)
    private Type type;

    @Transient
    private String content;

    @CreationTimestamp
    private Date creationDate;

    @UpdateTimestamp
    private Date lastModifiedDate;

    public Article(String title, Status status, Type type, String content) {
        this.title = title;
        this.status = status;
        this.type = type;
        this.content = content;
    }
}
```

```java
public enum Type {
    INTERNAL, EXTERNAL;
}
```

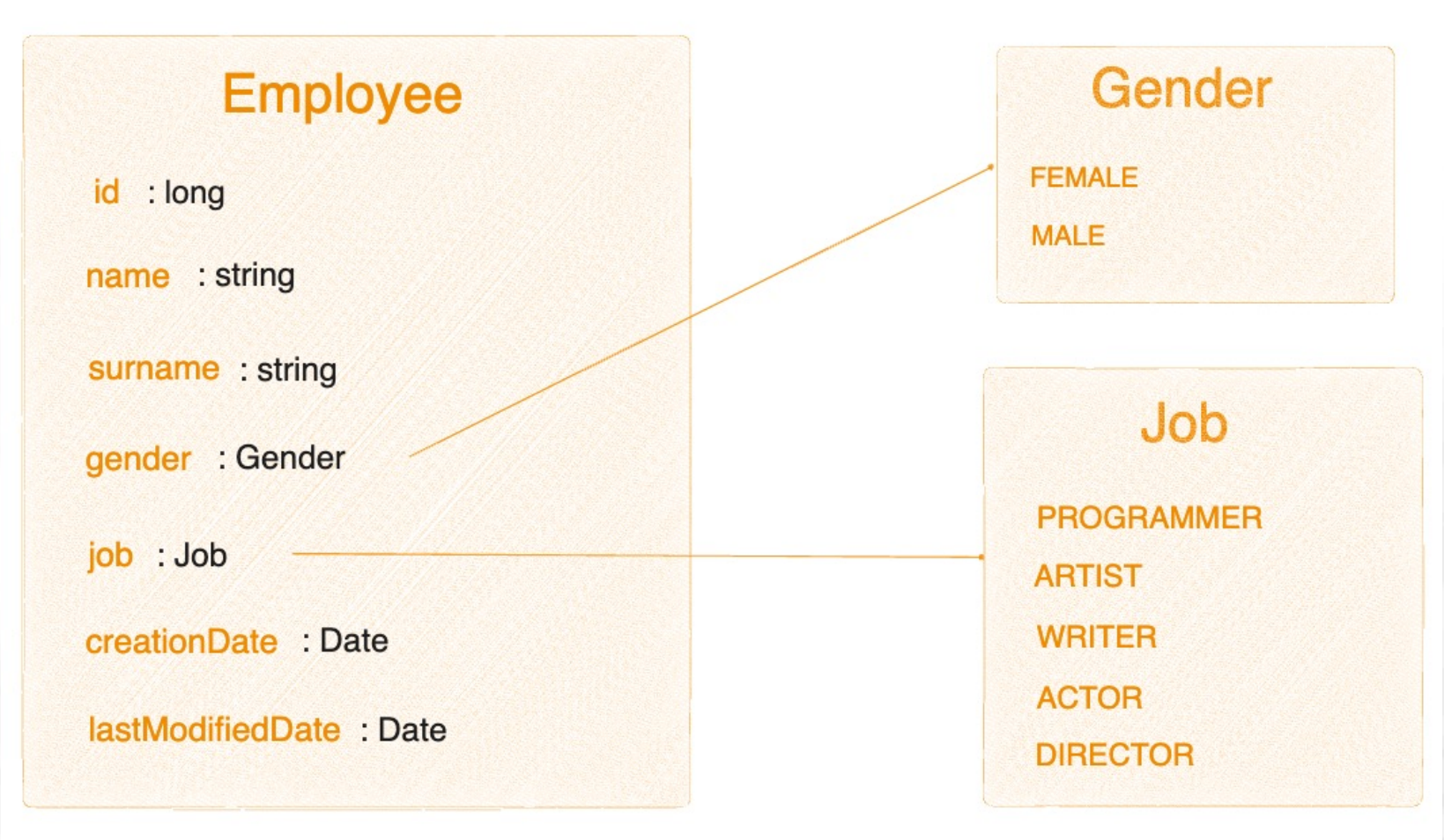
```java
public enum Status {
    OPEN,
    REVIEW,
    APPROVED,
    REJECTED;
}
```

```java
@Test
public void createArticleTest() {
    Article article = new Article( title: "Title", Status.APPROVED, Type.EXTERNAL);
    Article article2 = new Article( title: "Title2", Status.OPEN, Type.INTERNAL);
    Article article3 = new Article( title: "Title3", Status.REJECTED, Type.EXTERNAL);

    try(Session session = factory.openSession()) {
        Transaction tx = session.beginTransaction();
        session.persist(article);
        session.persist(article2);
        session.persist(article3);
        tx.commit();
    }
    try(Session session = factory.openSession()) {
        Article carFromDb = session.find(Article.class, article.getId());
        System.out.println(carFromDb);
    }
}
```

SELECT * FROM ARTICLE;

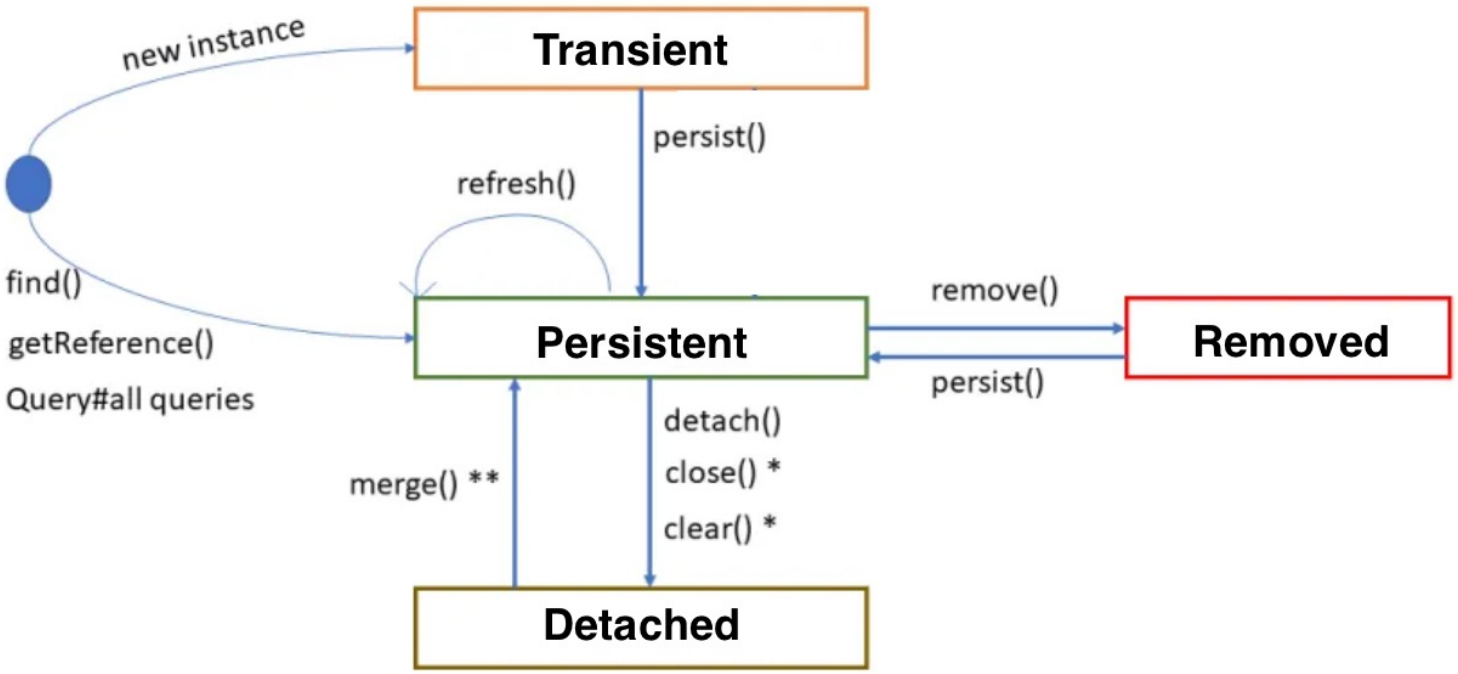
STATUS	CREATIONDATE	ID	LASTMODIFIEDDATE	TITLE	TYPE
2	2023-07-21 09:21:13.706	1	2023-07-21 09:21:13.706	Title	EXTERNAL
0	2023-07-21 09:21:13.72	2	2023-07-21 09:21:13.72	Title2	INTERNAL
3	2023-07-21 09:21:13.722	3	2023-07-21 09:21:13.722	Title3	EXTERNAL

ÖRNEK - 2


Employee
id : long
name : string
surname : string
gender : Gender
job : Job
creationDate : Date
lastModifiedDate : Date
Gender
FEMALE
MALE
Job
PROGRAMMER
ARTIST
WRITER
ACTOR
DIRECTOR

PERSISTENCE LIFECYCLE

PERSISTENCE LIFECYCLE



ÖRNEK -3 LIFECYCLE

ELEMENT COLLECTIONS

```java
@Entity
public class Person {
    @Id
    @GeneratedValue
    private Long id;

    private String name;

    private String surname;

    @ElementCollection
    @CollectionTable(name = "person_addresses")
    private List<String> addresses = new ArrayList<>();

    public Person() {}

    public Person(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

    public void addAddress(String address) {
        this.addresses.add(address);
    }

    public Long getId() {
        return id;
    }
}
```

LIST - SET

MAP

```java
@Entity
public class Product {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @ElementCollection
    @CollectionTable(name = "product_codes", joinColumns = @JoinColumn(name = "product_id"))
    @MapKeyColumn(name = "product_name")
    @Column(name = "product_code")
    private Map<String, String> productCodes = new HashMap<>();

    public Product(String name) {
        this.name = name;
    }

    public Product() {

    }

    public Map<String, String> getProductCodes() {
        return productCodes;
    }
}
```

```java
Product product = new Product( name: "Product A");
product.getProductCodes().put( k: "Product A1",  v: "ABC123");
product.getProductCodes().put( k: "Product A2",  v: "XYZ789");
product.getProductCodes().put( k: "Product A3",  v: "DEF456");

Product product2 = new Product( name: "Product B");
product2.getProductCodes().put( k: "Product B1",  v: "ABC123");
product2.getProductCodes().put( k: "Product B2",  v: "XYZ789");
product2.getProductCodes().put( k: "Product B3",  v: "DEF456");
```

SELECT * FROM PRODUCT ;

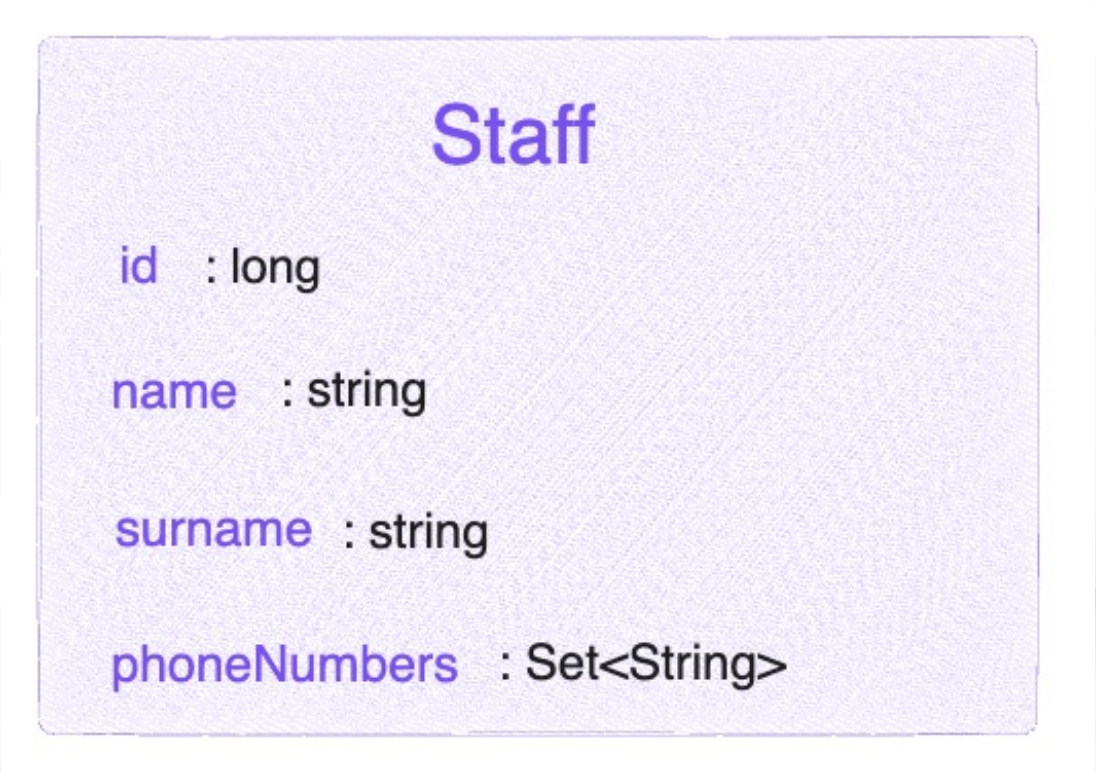
ID	NAME
1	Product A
2	Product B

(2 rows, 0 ms)

SELECT * FROM PRODUCT_CODES;

PRODUCT_ID	PRODUCT_CODE	PRODUCT_NAME
1	ABC123	Product A1
1	XYZ789	Product A2
1	DEF456	Product A3
2	ABC123	Product B1
2	XYZ789	Product B2
2	DEF456	Product B3

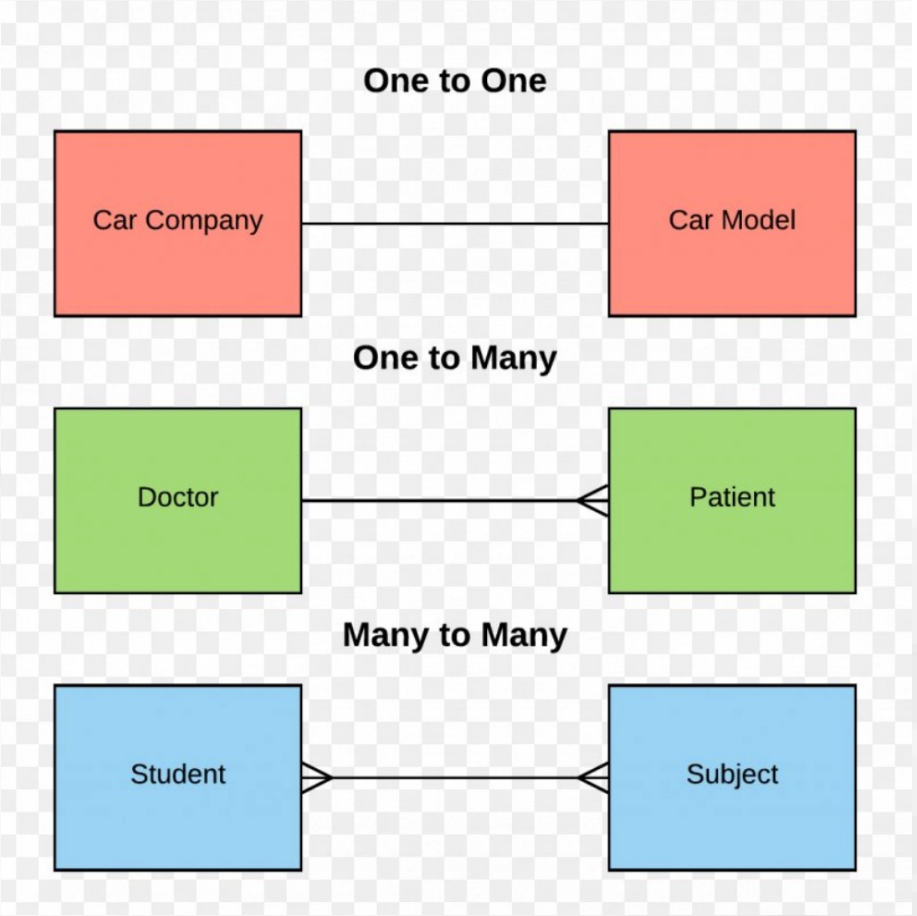
ÖRNEK -4


Staff
id : long
name : string
surname : string
phoneNumbers : Set<String>

ENTITY ASSOCIATIONS

Entity Relationships

- One to One (1:1)
- One to Many (1:M)
- Many to One (M:1)
- Many to Many (M:M)



ONE-TO-ONE

```java
@Entity(name = "users")
public class User {
    @Id
    @GeneratedValue
    private Long id;

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

    @OneToOne
    private UserDetail userDetail;


    public User() {
    }

    public User(String username) { this.username = username; }

    public void setUserDetail(UserDetail detail) {
        this.userDetail = detail;
    }

    public Long getId() {
        return id;
    }
}
```

```java
@Entity
public class UserDetail {

    @Id
    @GeneratedValue
    private Long id;

    private String email;
    private String address;
    private String number;

    @OneToOne
    private User user;

    public UserDetail() {
    }

    public UserDetail(String email, String address, String number, User user) {
        this.email = email;
        this.address = address;
        this.number = number;
        this.user = user;
    }
}
```

```java
@Test
public void createUserTest() {
    User user = new User( username: "Ezgi");
    UserDetail userDetail = new UserDetail( email: "e@gmail.com",  address: "Ankara, Turkey",  number: "123456", user);
    userDetail.setUser(user);

    try(Session session = factory.openSession()) {
        Transaction tx = session.beginTransaction();
        session.persist(userDetail);
        session.persist(user);
        tx.commit();
    }
    try(Session session = factory.openSession()) {
        User userFromDb = session.find(User.class, user.getId());
        assertNotNull(userFromDb.toString());
    }
}
```

SELECT * FROM USERS ;

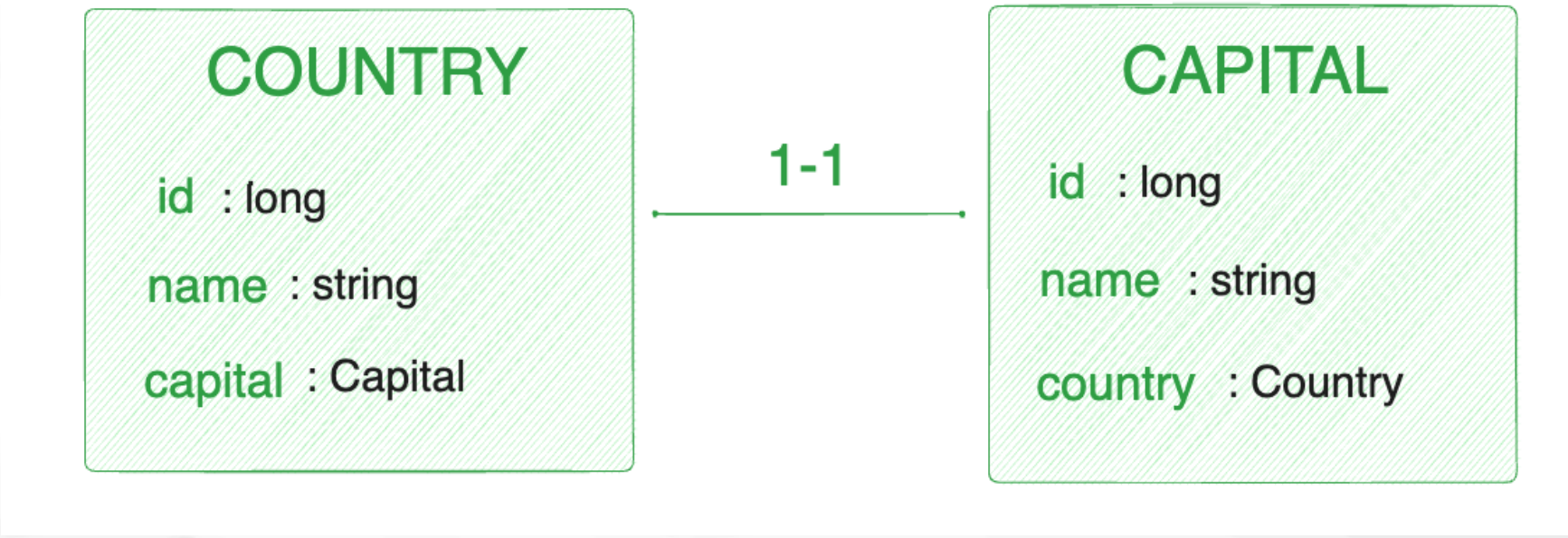
ID	USERDETAIL_ID	NAME
1	1	Ezgi

(1 row, 1 ms)

SELECT * FROM USERDETAIL;

ID	USER_ID	ADDRESS	EMAIL	NUMBER
1	1	Ankara, Turkey	e@gmail.com	123456

ÖRNEK-5 ONE-TO-ONE



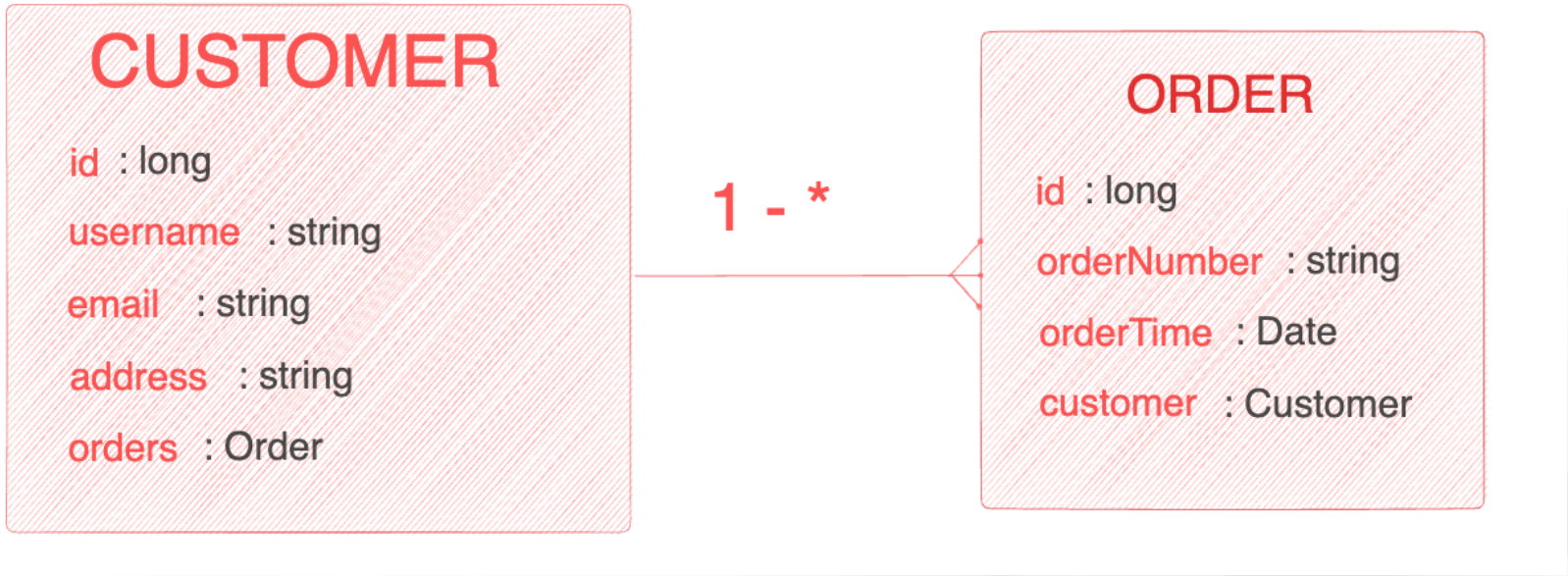
ONE-TO-MANY

```java
@Getter
@Setter
@NoArgsConstructor
@Entity
public class Project {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String description;

    @Enumerated(EnumType.STRING)
    private ProjectStatus status;

    @OneToMany
    private Set<Task> tasks = new HashSet<>();

    public Project(String name, String description, ProjectStatus status) {
        this.name = name;
        this.description = description;
        this.status = status;
    }

    public void addTask(Task task) {
        this.tasks.add(task);
    }
}
```

```java
@Entity
public class Task {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String description;

    @Temporal(TemporalType.DATE)
    private LocalDate startDate;

    @Temporal(TemporalType.DATE)
    private LocalDate endDate;

    public Task() {
    }

    public Task(String description, LocalDate startDate, LocalDate endDate) {
        this.description = description;
        this.startDate = startDate;
        this.endDate = endDate;
    }

    public Long getId() {
        return id;
    }
}
```

```java
@Test
public void createProjectTest() {
    Project project = new Project( name: "Yeni proje",  description: "önemli", ProjectStatus.NEW);
    Project project2 = new Project( name: "2.proje",  description: "acil değil", ProjectStatus.IN_PROGRESS);

    Task task = new Task( description: "task1", LocalDate.now(), LocalDate.now().plusMonths( monthsToAdd: 4));
    Task task2 = new Task( description: "task2", LocalDate.now().plusDays( daysToAdd: 4), LocalDate.now().plusMonths( monthsToAdd: 2));
    Task task3 = new Task( description: "task3", LocalDate.now(), LocalDate.now().plusMonths( monthsToAdd: 3));

    project.addTask(task);
    project.addTask(task2);
    project2.addTask(task3);

    try(Session session = factory.openSession()) {
        Transaction tx = session.beginTransaction();
        session.persist(task);
        session.persist(task2);
        session.persist(task3);
        session.persist(project);
        session.persist(project2);
        tx.commit();
    }

}
```

ÖRNEK -6 ONE-TO-MANY



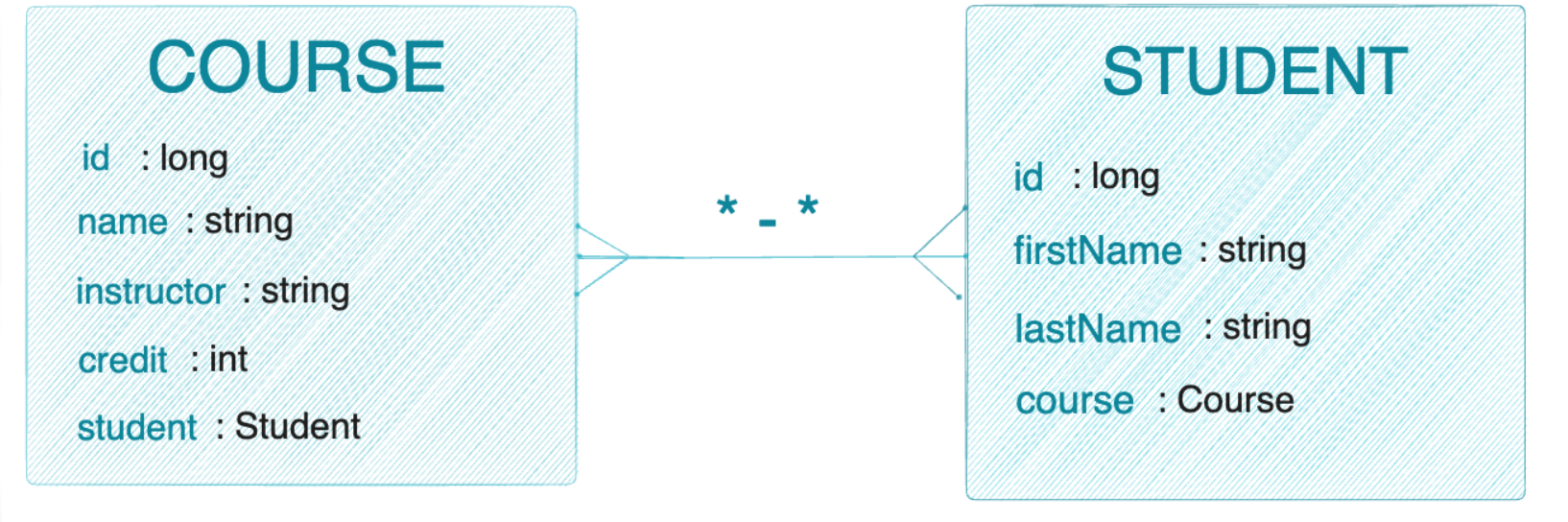
MANY-TO-MANY

```java
@NoArgsConstructor
@Getter
@Entity
public class Author {
    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @ManyToMany
    @JoinTable(name = "book_author",
            joinColumns = @JoinColumn(name = "author_id"),
            inverseJoinColumns = @JoinColumn(name = "book_id"))
    private Set<Book> books = new HashSet<>();

    public Author(String name) {
        this.name = name;
    }

    public void addBook(Book book) {
        this.books.add(book);
    }
}
```

```java
@ToString
@Getter
@NoArgsConstructor
@Entity(name = "Book")
public class Book {
    @Id
    @GeneratedValue
    private Long id;

    private String title;

    @ManyToMany(mappedBy = "books")
    private Set<Author> authors = new HashSet<>();

    public Book(String title) {
        this.title = title;
    }
}
```

```java
@Test
public void createBookTest() {
    Book book = new Book( title: "book1");
    Author author = new Author( name: "author1");
    author.addBook(book);
    try(Session session = factory.openSession()) {
        Transaction tx = session.beginTransaction();
        session.persist(book);
        session.persist(author);
        tx.commit();
    }

    try (Session session = factory.openSession()) {
        Book bookFromDB = session.find(Book.class, book.getId());
        System.out.println(bookFromDB);
        assertEquals(bookFromDB.getAuthors().size(), expected: 1);
    }
}
```

ÖRNEK-7 MANY TO MANY



Cascade Types

Cascade, bir JPA standardıdır. Entity sınıflarımızdaki ilişkilerin hareketlerini yani davranışlarını cascade tipleri ile ayarlarız. Yani ilişkili sınıfların birbirlerinden etkilenip etkilenmemesini sağlıyor.

Örnek olarak bir değer sildiğimizde o silinen veri ilişkili olan verilerin etkilenmesini ya da etkilenmemesini sağlarız.

Cascade tipleri nelerdir?

TİP	Görevi
Persist	Nesne persist edilirse ilişkili nesnelerde persist edilir
All	Tüm işlemleri ilişkili nesnelerle birlikte yapar
Merge	Nesne merge edilirse ilişkili nesnelerde merge edilir
Remove	Nesne remove edilirse ilişkili nesnelerde remove edilir
Refresh	Nesne refresh edilirse ilişkili nesnelerde refresh edilir

Fetch Types

- Aralarında ilişki bulunan Entity sınıflarından bir tarafın yüklenme durumunda diğer tarafın yüklenme stratejisini belirlememizi sağlar. Hibernate de 2 adet fetch type vardır. Bunlar:,

    1-) Eager(Ön Yükleme)

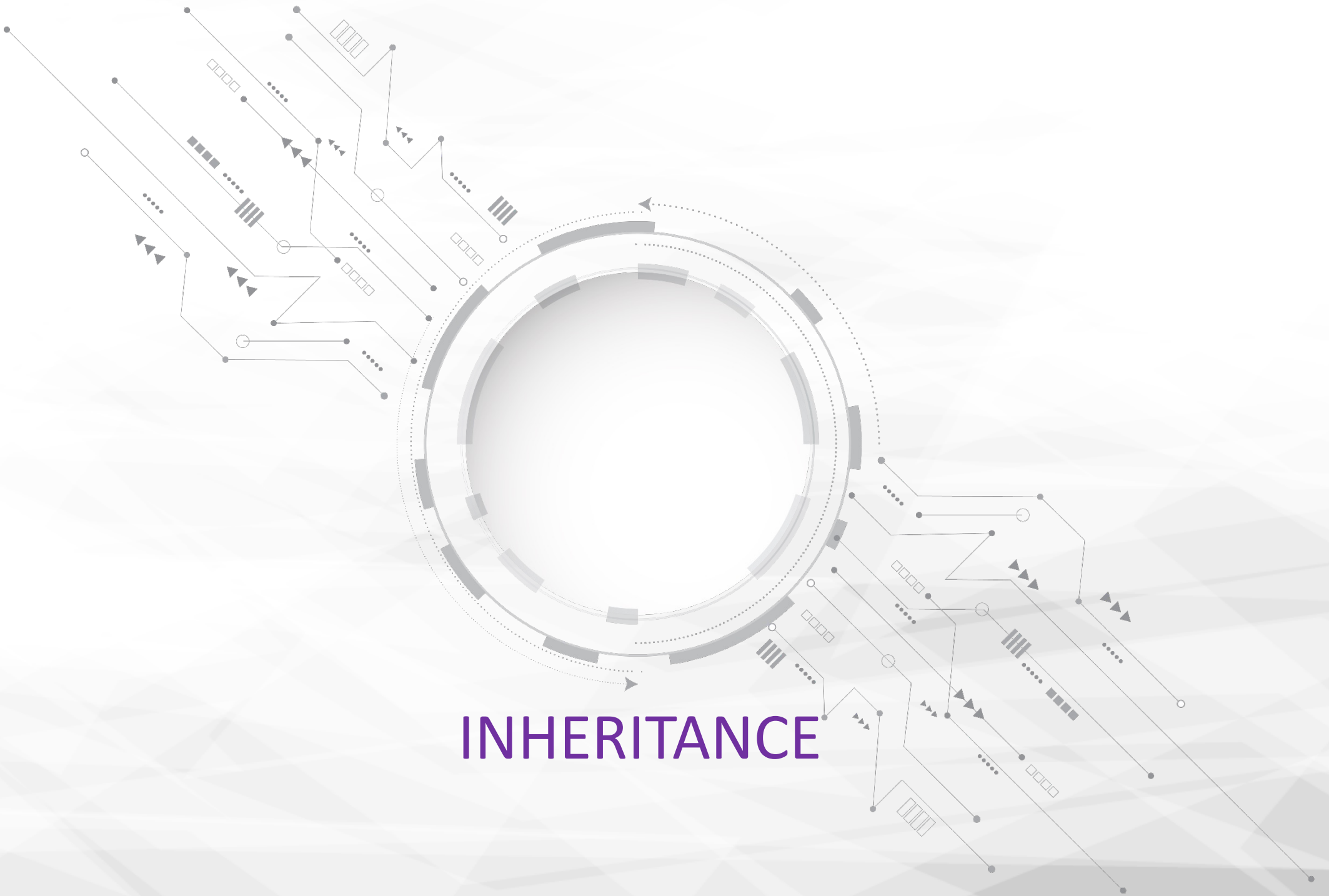
    2-) Lazy(Tembel/Sonradan Yükleme)

- OneToOne veya ManyToOne → FetchType.EAGER
- OneToMany veya ManyToMany → FetchType.LAZY

```java
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    no usages
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")
    private List<Order> orders = new ArrayList<>();
}
```

```java
@Entity
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    no usages
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "customer_id")
    private Customer customer;
}
```

QUERY



INHERITANCE

TÜBİTAK
BİLGEM

TEŞEKKÜR EDERİZ