Exercise: Hibernate Relations Cheat Sheet

This document defines a **cheat sheet for relations** in Hibernate for the "Databases Advanced – Hibernate" course @ Software University.

There are 2 types of relationships in Hibernate **unidirectional** and **bidirectional.**

Unidirectional means that the relationship is valid in **only one direction**. For example, an operating system is installed on a computer. A computer is not installed on an operating system.

Bidirectional means that the relationship is valid in **both directions**. For example a person owns a phone and the phone is owned by a person.

Unidirectional

1. One-to-Many

The employee knows who is his employer but the employer does not know who are his employees.

Employer = parent table **Employee** = child table

Employee.java

```
@Entity
public class Employee {
    private Long id;
    private Employer employer;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

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

    @ManyToOne
    public Employer getEmployer() {
        return employer;
    }

    public void setEmployer(Employer employer) {
        this.employer = employer;
    }
}
```

2. One-to-One

The employee does not know his address but the address knows who employee lives on it.

Employee = parent table **Address** = child table

Address.java

```
@Entity
public class Address {
    private Long employeeId;
    private Employee employee;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(Long employeeId) {
        this.employeeId = employeeId;
    }

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "employee_id")
    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }
}
```

3. Many-to-Many

One book can be written by many authors but the authors do not know which books they published.

Author = parent table **Book** = child table

Book.java

```
@Entity
public class Book {
    private Long bookId;
    private Set<Author> authors;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getBookId() {
        return bookId;
    }

    public void setBookId(Long bookId) {
        this.bookId = bookId;
    }

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(
            name = "author_book",
            joinColumns = @JoinColumn(name = "book_id"),
            inverseJoinColumns = @JoinColumn(name = "author_id"))
    public Set<Author> getAuthors() {
        return authors;
    }

    public void setAuthors(Set<Author> authors) {
        this.authors = authors;
    }
}
```

Bidirectional

1. One-to-Many

The employee knows who is his employer and the employer know who are his employees.

Employer = parent table **Employee** = child table

Employee.java

```
@Entity
public class Employee {
    private Long id;
    private Employer employer;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

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

    @ManyToOne(cascade = CascadeType.ALL)
    public Employer getEmployer() {
        return employer;
    }

    public void setEmployer(Employer employer) {
        this.employer = employer;
    }
}
```

Employer.java

```
@Entity
public class Employer {
    private Long id;
    private Set<Employee> employees;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

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

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "employer")
    public Set<Employee> getEmployees() {
        return employees;
    }

    public void setEmployees(Set<Employee> employees) {
        this.employees = employees;
    }
}
```

2. One-to-One

An employee lives on exactly one address and the address has reference to the employee that lives there.

Employee = parent table **Address** = child table

Address.java

```
@Entity
public class Address {
    private Long employeeId;
    private Employee employee;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(Long employeeId) {
        this.employeeId = employeeId;
    }

    @OneToOne(cascade = CascadeType.ALL)
```

```java
    @JoinColumn(name = "employee_id")
    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }
}
```

Employee.java

```java
@Entity
public class Employee {
    private Long id;
    private Address address;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "employee_id")
    public Long getId() {
        return id;
    }

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

    @OneToOne(cascade = CascadeType.ALL, mappedBy = "employee")
    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }
}
```

3. Many-to-Many

A book can be written by many authors and many authors can write single book.

Author = parent table **Book** = child table

Book.java

```java
@Entity
public class Book {
    private Long bookId;
    private Set<Author> authors;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "book_id")
    public Long getBookId() {
        return bookId;
    }

    public void setBookId(Long bookId) {
        this.bookId = bookId;
    }

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(
            name = "author_book",
            joinColumns = @JoinColumn(name = "book_id"),
            inverseJoinColumns = @JoinColumn(name = "author_id"))
    public Set<Author> getAuthors() {
        return authors;
    }

    public void setAuthors(Set<Author> authors) {
        this.authors = authors;
    }
}
```

Author.java

```java
@Entity
public class Author {
    private Long authorId;
    private Set<Book> books;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "author_id")
    public Long getAuthorId() {
        return authorId;
    }

    public void setAuthorId(Long authorId) {
        this.authorId = authorId;
    }

    @ManyToMany(cascade = CascadeType.ALL, mappedBy = "authors")
    public Set<Book> getBooks() {
        return books;
    }

    public void setBooks(Set<Book> books) {
        this.books = books;
    }
}
```