JAVA TECHNOLOGY

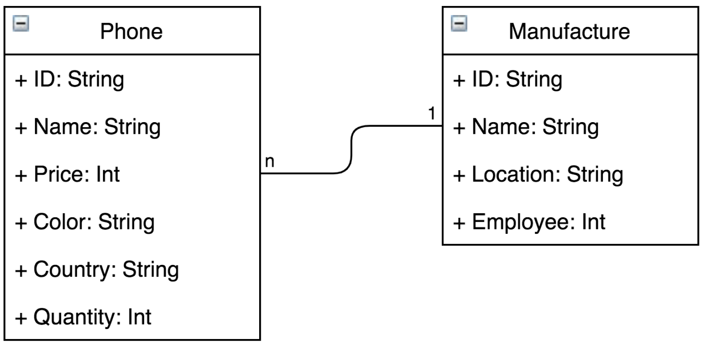
(503111)

LAB 3

LAB OBJECTIVES

Use Hibernate - a Java framework that simplifies tasks when working with the database system in Java applications. This is one of the most commonly used open source ORM (Object Relational Mapping) tools in java applications. In this lab, students will learn how to use Hibernate to connect and interact in an object-oriented style with a database system to perform basic tasks such as adding, deleting, updating and reading data.

Given two classes Phone and Manufacture represents the phone entity and manufacturer entity as shown below.

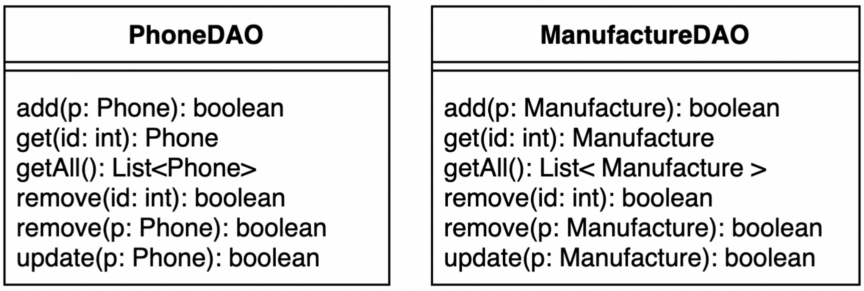


Requirements:

1. Create POJO (Plain Old Java Object) classes based on class diagram description above.

- The Manufacture class contains a list of Phone objects of type List<Phone>. The Phone class has many properties, one of which is of type Manufacture.
- Configure (with Hibernate annotations) so that the names of the two tables created in the database are the MobilePhone and Manufacture tables.
- Set the constraints so that the MobilePhone table has ID as its primary key. The Name, Price and Color columns are required columns (not null).
- Set the constraints so that the Manufacture table has ID as its primary key. The employee attribute (int) in this table indicates the number of employees of the respective manufacturer.
- Set the constraints so that the phone name and manufacturer name columns accept string values between 3 and 128 characters.

2. Create a PhoneDAO class that provides the CRUD operations (create, read, update and delete) on the Phone table.
3. Create the ManufactureDAO class that provides a CRUD operation on the Manufacture table. The class diagram of the two DAO classes at this time has the following form:



```
PhoneDAO
-------------------------------
add(p: Phone): boolean
get(id: int): Phone
getAll(): List<Phone>
remove(id: int): boolean
remove(p: Phone): boolean
update(p: Phone): boolean
```

```
ManufactureDAO
-------------------------------
add(p: Manufacture): boolean
get(id: int): Manufacture
getAll(): List< Manufacture >
remove(id: int): boolean
remove(p: Manufacture): boolean
update(p: Manufacture): boolean
```

4. Continue adding the following methods to the PhoneDAO class:

- A method that returns the phone with the highest selling price.
- A method to get a list of phones sorted by country name, if two phones have the same country, sort descending by price.
- A method to check if there is a phone priced above 50 million VND.
- A method to get the first phone in the list that meets the criteria: has the color 'Pink' and costs over 15 million. If there are no matching phones, return null.

5. Continue adding the following methods to the ManufactureDAO class.
   - A method to check whether all manufacturers have more than 100 employees.
   - A method that returns the sum of all employees of the manufactures.
   - A method that returns the last manufacturer in the list of manufacturers that meet the criteria: based in the US. If there is no producer that meets the above criteria, throw an InvalidOperationException.

Create a **Program.java** file containing the main method to call all the functions you just wrote to test, creating a complete application that works with Phone and Manufacture objects.