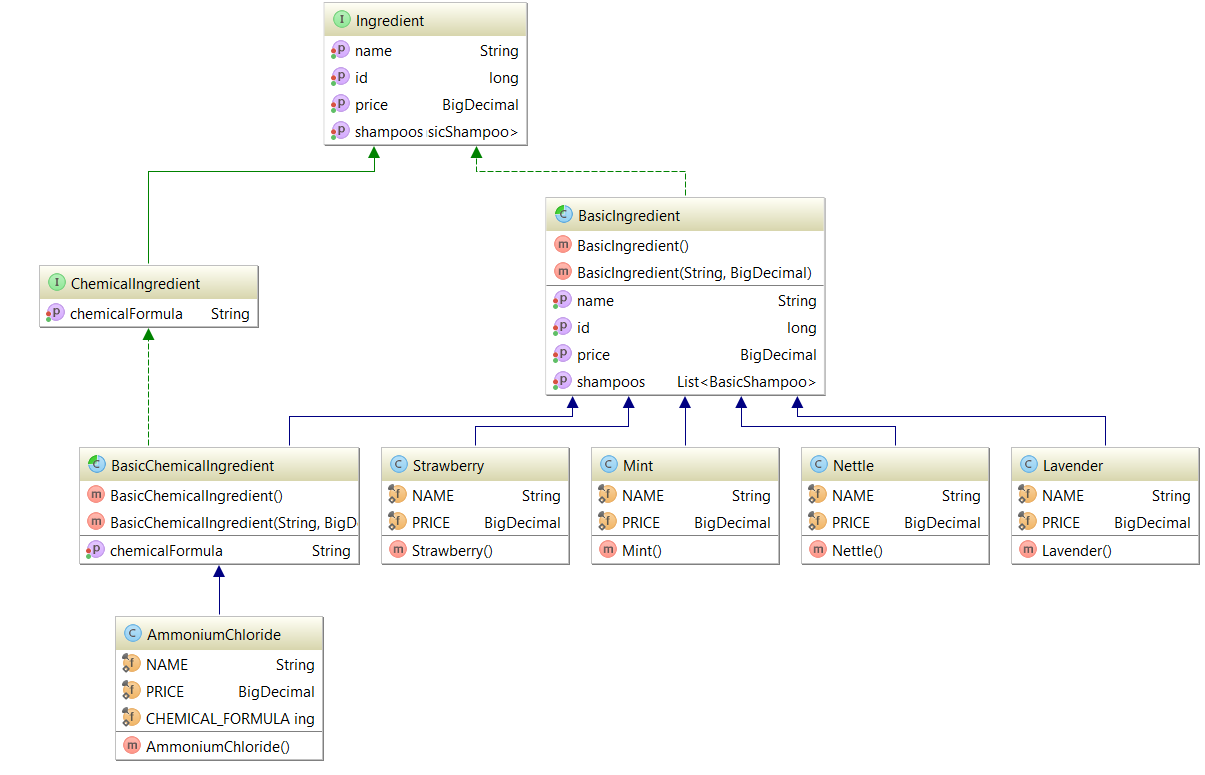
# Exercises: Shampoo Company

This document defines the **exercise assignments** for the ["Databases Advanced – Hibernate" course @ Software University.](https://softuni.bg/trainings/1444/databases-advanced-hibernate-october-2016)

We are given a task to create the back-end of the new Shampoo Company “Neck and Elbows”. Here is what you have to accomplish:

## Create Ingredients

Every shampoo has ingredients. Here is the expected structure of the ingredients:



Define two interfaces as follows:

**public interface** Ingredient **extends** Serializable{  
  
 **long** getId();  
  
 **void** setId(**long** id);  
  
 String getName();  
  
 **void** setName(String name);  
  
 BigDecimal getPrice();  
  
 **void** setPrice(BigDecimal price);  
}

**public interface** ChemicalIngredient **extends** Ingredient{  
  
 String getChemicalFormula();  
  
 **void** setChemicalFormula(String chemicalFormula);  
}

There are two types of ingredients:

* **Basic Ingredient**. It has the following information:
  + **Id**
  + **Name**
  + **Price**
* **Chemical Ingredient** which add additional information
  + **Chemical formula**

We have 4 types of Basic Ingredients:

* **Mint**
  + Price 3.54
* **Nettle**
  + Price 6.12
* **Strawberry**
  + Price 4.85
* **Lavender**
  + Price 2

And only one Chemical Ingredient:

* **Ammonium Chloride**
  + Price 0.59
  + Formula NH4Cl

**Create the required classes.**

## Implement the inheritance in the database

**All the ingredients should be saved in MySQL as a Single Table. Use the following strategy:**

@Inheritance(strategy = InheritanceType.***SINGLE\_TABLE***)  
@DiscriminatorColumn(name = **"type"**, discriminatorType = DiscriminatorType.***STRING***)

## Create Enumerator Size

**Create an enumerator Size with the following constants:**

***SMALL***, ***MEDIUM***, ***BIG***;

## Create Label

Implement the following interface:

**public interface** Label **extends** Serializable {  
  
 **long** getId();  
  
 **void** setId(**long** id);  
  
 String getTitle();  
  
 **void** setTitle(String title);  
  
 String getSubTitle();  
  
 **void** setSubTitle(String subTitle);  
}

We need to create one class Classic Label with the following fields:

* **ClassicLabel**
  + **Id**
  + **Title**
  + **Subtitle**

**Create the class.**

## Create Batch

Implement the following interface:

**public interface** Batch **extends** Serializable{  
  
 **long** getId();  
  
 **void** setId(**long** id);  
  
 Date getBatchDate();  
  
 **void** setBatchDate(Date batchDate);  
  
 Set<BasicShampoo> getShampoos();  
  
 **void** setShampoos(Set<BasicShampoo> shampoos);  
}

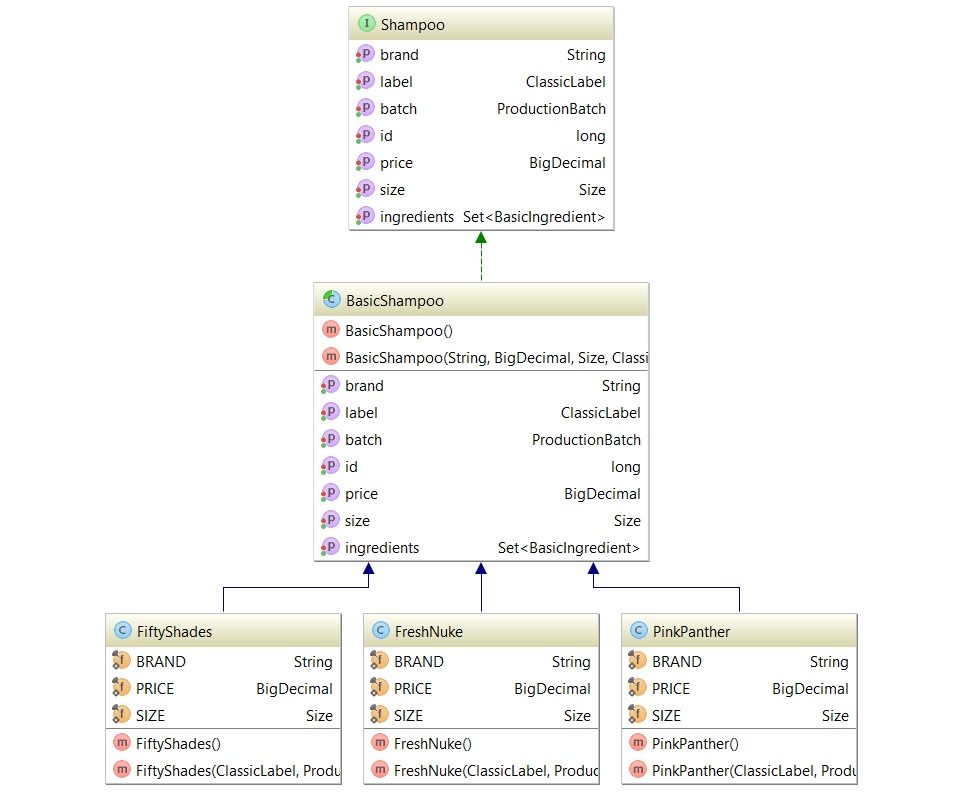
We need a class Production Batch that will hold set of shampoos.

* **ProductionBatch**
  + **Id**
  + **Batch Date**
  + **Shampoos**

**Create the class.**

## Create Shampoos

It is required to create the following structure:



Implement the following Interface:

**public interface** Shampoo **extends** Serializable{  
  
 **long** getId();  
  
 **void** setId(**long** id);  
  
 String getBrand();  
  
 **void** setBrand(String brand);  
  
 BigDecimal getPrice();  
  
 **void** setPrice(BigDecimal price);  
  
 Size getSize();  
  
 **void** setSize(Size size);  
  
 ClassicLabel getLabel();  
  
 **void** setLabel(ClassicLabel label);  
  
 ProductionBatch getBatch();  
  
 **void** setBatch(ProductionBatch batch);  
  
 Set<BasicIngredient> getIngredients();  
  
 **void** setIngredients(Set<BasicIngredient> ingredients);  
}

We need an abstract class BasicShampoo that hold the following information:

* **BasicShampoo**
  + **Id**
  + **Brand**
  + **Price**
  + **Size**
  + **Label**
  + **Batch**
  + **Set of Ingredients**

We have 4 different types of Shampoos:

* **Fresh Nuke**
  + **Brand “Fresh Nuke”**
  + **Price 9.33**
  + **Size Big**
  + **Label with title "Fresh Nuke" and subtitle "Explosively cool"**
  + **It’s made of Mint, Nettle and Ammonium Chloride**
* **Pink Panther**
  + **Brand “Pink Panther”**
  + **Price 8.50**
  + **Size Medium**
  + **Label with title "Pink Panther" and subtitle "Pleasure in pink "**
  + **It’s made of Lavender and Nettle**
* **Fifty Shades**
  + **Brand “Fifty Shades”**
  + **Price 6.69**
  + **Size Small**
  + **Label with title "Fifty Shades" and subtitle "Tie the aroma"**
  + **It’s made of Strawberry and Nettle**

**Create the classes. The inheritance should be presented in a single table.**

## Persistent enumerator

**Make the field size in BasicShampoo persistent. You can use the following annotation:**

@Enumerated(value = EnumType.***STRING***)

## One-To-One Relation

**Create a unidirectional relation between Shampoo and Label which is type one-to-one:**

@OneToOne(optional = **false**)  
@JoinColumn(name = **"label\_id"**, referencedColumnName = **"id"**)

## One-To-Many/Many-To-One Relation

**Create a bidirectional relation between Shampoo and Batch:**

@ManyToOne(optional = **false**)  
@JoinColumn(name = **"batch\_id"**, referencedColumnName = **"id"**)  
**private** ProductionBatch **batch**;

@OneToMany(mappedBy = **"batch"**, targetEntity = BasicShampoo.**class**)  
**private** Set<Shampoo> **shampoos**;

## Many-To-Many Relation

Add one more property to our interface Ingredient:

Set<BasicShampoo> getShampoos();  
  
**void** setShampoos(Set<BasicShampoo> shampoos);

Every ingredient should have a list of shampoos.

**Create a bidirectional relation between Shampoo and Ingredient:**

@ManyToMany  
@JoinTable(name = **"shampoos\_ingredients"**,  
joinColumns = @JoinColumn(name = **"shampoo\_id"**, referencedColumnName = **"id"**),  
inverseJoinColumns = @JoinColumn(name = **"ingredient\_id"**, referencedColumnName = **"id"**))  
**private** Set<BasicIngredient> **ingredients**;

@ManyToMany(mappedBy = **"ingredients"**, targetEntity = BasicShampoo.**class**)  
**private** List<BasicShampoo> **shampoos**;

## Repositories

**Create repository for every entity we have.**

## Services

**Create service for every repository we have.**

## Create Objects

**Create one shampoo of each type and save it to the database.**