Exercise: Hibernate Code First - Shampoo Company

This document defines the lab assignments for the "Databases Frameworks" course at Software University.

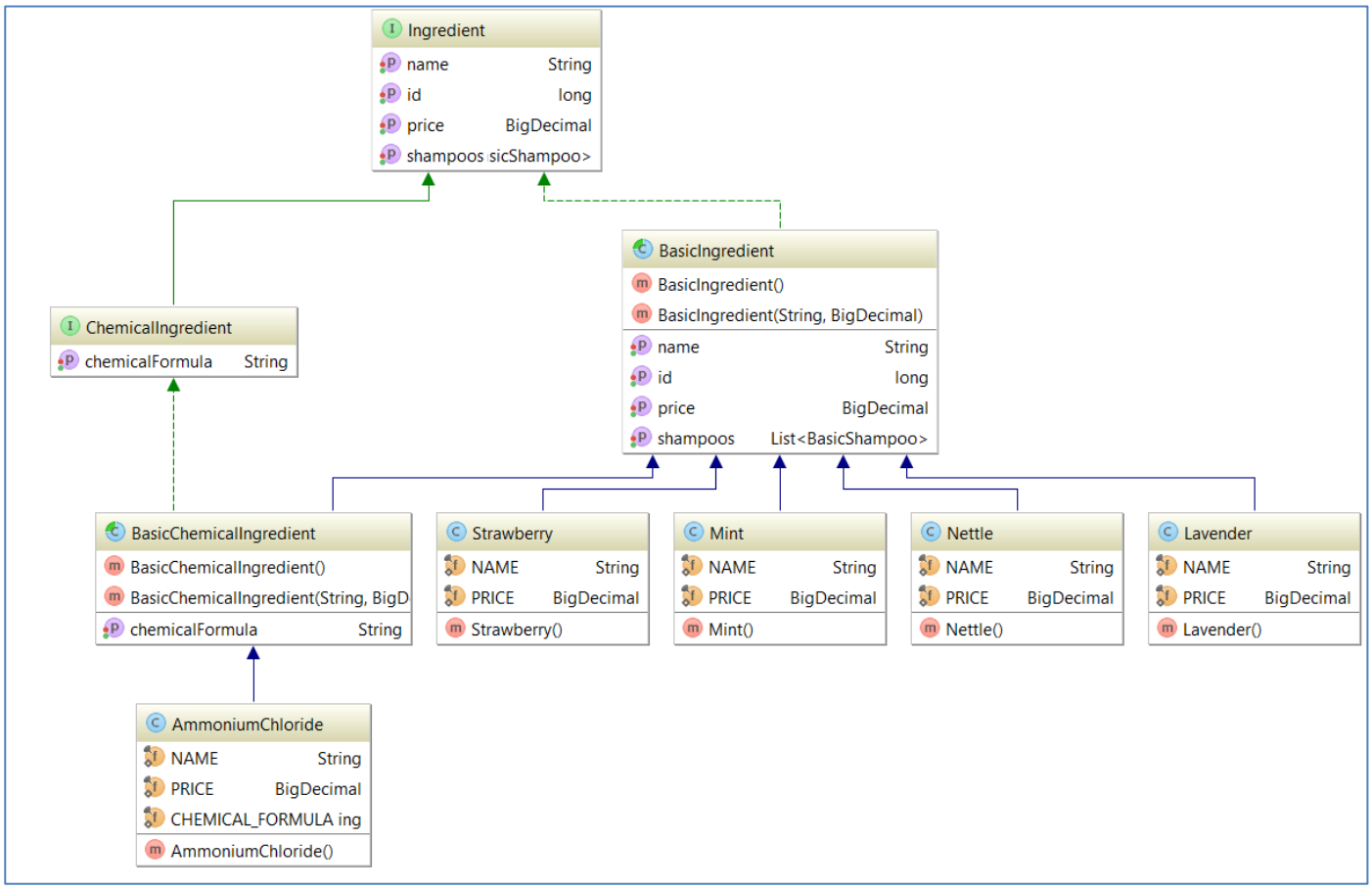
Your task is to create a shampoo company hierarchy. The company produces shampoos, which have **certain set of ingredients**, **label**, **price** and **brand**.

There are several types of shampoos:

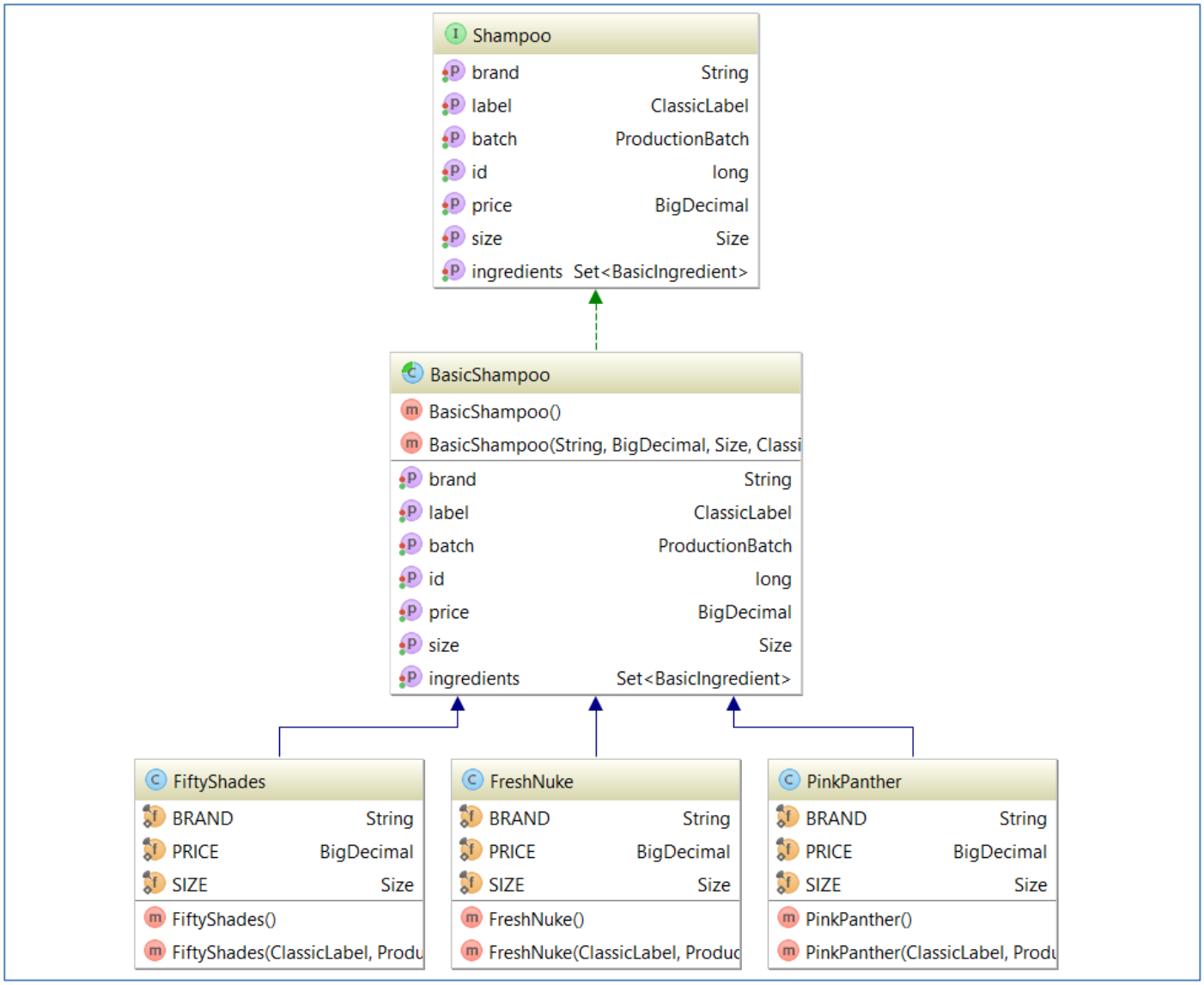
- **Fifty shade**
  - brand: "Fifty Shades"
  - price: 6.69
- **FreshNuke**
  - brand: "Fresh Nuke"
  - price: 9.33
- **Pink Panther**
  - brand: "Pink Panther"
  - Price: 8.50

Each shampoo can be either size **`BIG`**, **`SMALL`** or **`MEDIUM`.**

The ingredient structure should look like this:



The shampoo structure should look like this:



1. Hierarchy Setup

Start by creating some interfaces for further implementation:

Shampoo:

```
public interface Shampoo {

    long getId();

    void setId(long id);

    String getBrand();

    void setBrand(String brand);

    BigDecimal getPrice();

    void setPrice(BigDecimal price);

    Size getSize();

    void setSize(Size size);

    BasicLabel getLabel();

    void setLabel(BasicLabel label);

    Set<BasicIngredient> getIngredients();

    void setIngredients(Set<BasicIngredient> ingredients);
}
```

Ingredient:

```
interface Ingredient extends Serializable {

    String getName();

    void setName(String name);

    int getId();

    void setId(int id);

    BigDecimal getPrice();

    void setPrice(BigDecimal price);

    List<BasicShampoo> getShampoos();

    void setShampoos(List<BasicShampoo> shampoos);
}
```

ChemicalIngredient:

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

Label:

```
public interface Label extends Serializable {
    long getId();

    void setId(long id);

    String getTitle();

    void setTitle(String title);

    String getSubtitle();

    void setSubtitle(String subtitle);
}
```

2. Ingredients

Create root classes **BasicIngredient** and **ChemicalIngredient**, which will be extended by concrete classes later.

All of our ingredients will be **stored in a single table** – "**ingredients**". They will be discriminated only by their **type.**

There are **2 types** of ingredients:

- **Basic Ingredient**. It has the following information:
  - **Id**
  - **Name**
  - **Price**
- **Chemical Ingredient**, which adds additional information:
  - **Chemical formula**

BasicIngredient:

```java
@Entity
@Table(name = "ingredients")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "ingredient_type", discriminatorType = DiscriminatorType.STRING)
public abstract class BasicIngredient implements Ingredient {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

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

    @Column(name = "price")
    private BigDecimal price;

    @ManyToMany(mappedBy = "ingredients", cascade = CascadeType.ALL)
    private List<BasicShampoo> shampoos;

    protected BasicIngredient(){};
    protected BasicIngredient(String name, BigDecimal price){
        this.name = name;
        this.price = price;
    }

    // Getters and setters
```

ChemicalIngredient:

```java
@Entity
public abstract class BasicChemicalIngredient extends BasicIngredient
        implements ChemicalIngredient {
    @Column(name = "chemical_formula")
    String chemicalFormula;

    protected BasicChemicalIngredient() {
    }

    BasicChemicalIngredient(String name, BigDecimal price, String chemicalFormula) {
        super(name, price);
        this.setChemicalFormula(chemicalFormula);
    }
}
```

Implement **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

Strawberry Ingredient:

```
@Entity
@DiscriminatorValue(value = "ST")
public class Strawberry extends BasicIngredient{

    private static final String NAME = "Strawberry";

    private static final BigDecimal PRICE = new BigDecimal( val: "4.85");

    public Strawberry() { super(NAME, PRICE); }
}
```

Ammonium Chloride:

```
@Entity
@DiscriminatorValue(value = "AM")
public class AmmoniumChloride extends BasicChemicalIngredient{

    private static final BigDecimal PRICE = new BigDecimal( val: "6.12");

    private static final String NAME = "Ammonium Chloride";

    private static final String CHEMICAL_FORMULA = "NH4Cl";

    public AmmoniumChloride() { super(NAME,PRICE,CHEMICAL_FORMULA); }

    // Getters and setters
```

Create the other ingredient classes analogically.

3. Label

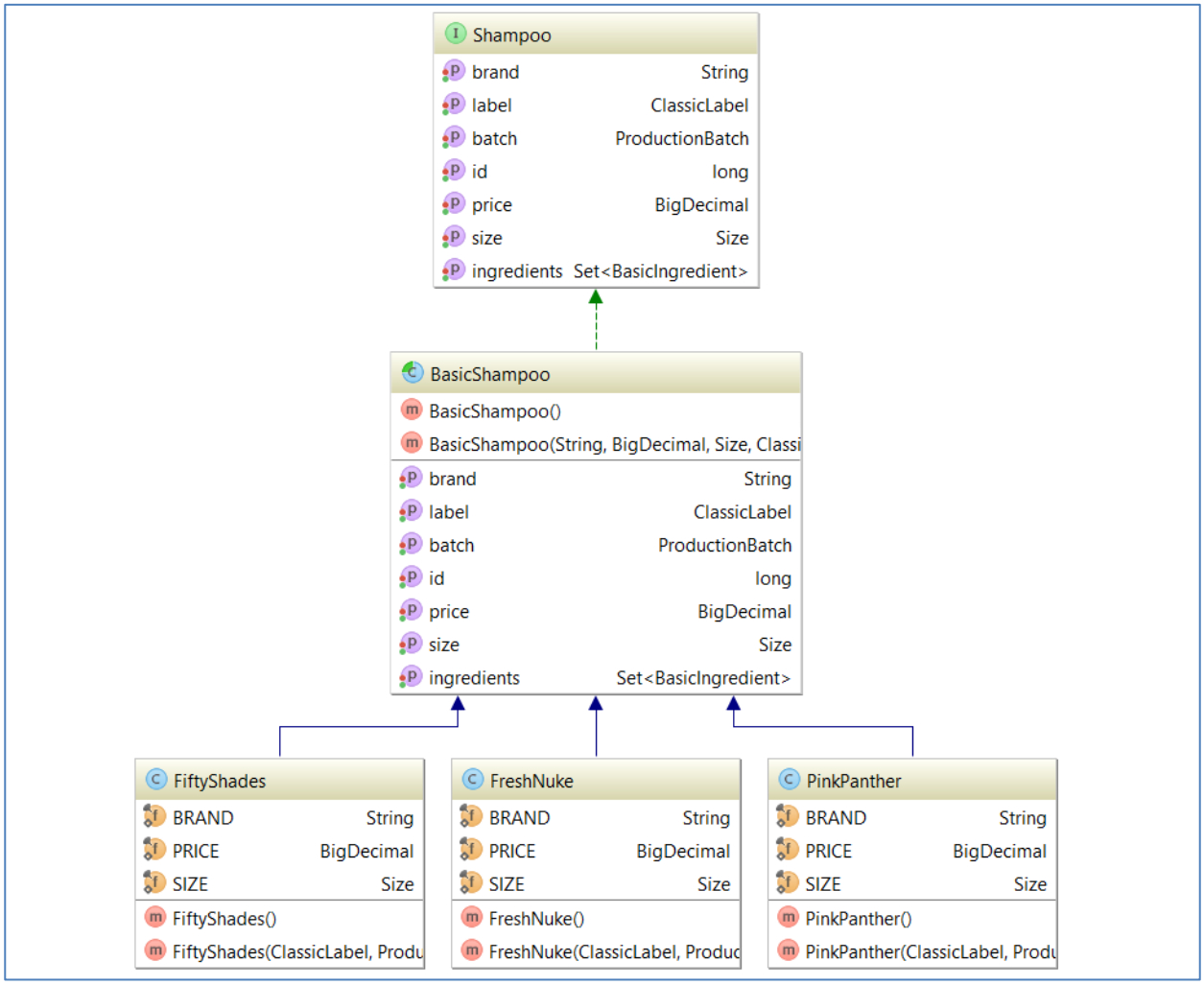
Create an abstract implementation of the Label interface you've created earlier.Each label will have the following fields: **id, title, subtitle.**

BasicLabel:

```java
@Entity
@Table(name = "labels")
public class BasicLabel implements Label {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Basic
    private String title;

    @Basic
    private String subtitle;

    @OneToOne(mappedBy = "label", targetEntity = BasicShampoo.class, cascade = CascadeType.ALL)
    private BasicShampoo basicShampoo;

    public BasicLabel(){};
    public BasicLabel(String title, String subtitle){
        this.title = title;
        this.subtitle = subtitle;
    }
    // Getters and setters
```

4. Create Shampoos

It is required to create the following structure:



Implement the Shampoo interface you've created earlier, by adding a new **root** class:

```
@Entity
@Table(name = "shampoos")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "shampoo_type", discriminatorType = DiscriminatorType.STRING)
public class BasicShampoo implements Shampoo {
    @Id
    private long id;

    @Basic
    private BigDecimal price;

    @Basic
    private String brand;

    @Enumerated
    private Size size;

    protected BasicShampoo(){this.setIngredients(new HashSet<>());}
    BasicShampoo(String brand, BigDecimal price, Size size, BasicLabel classicLabel) {...}
```

Each shampoo will hold information about it's label and ingredients. We will implement that using **table relations**. Add 2 new fields to the abstract implementation:

```
@OneToOne(optional = true, cascade = CascadeType.ALL,
fetch = FetchType.LAZY)
@JoinColumn(name = "label", referencedColumnName = "id")
private BasicLabel label;
```

Every shampoo's label will have **unidirectional One-to-One** relationship with a Label entity, mapped by the **`id` column of the label** and the **`label` field in the shampoo implementation**.

```
@ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinTable(name = "shampoos_ingredients",
joinColumns = @JoinColumn(name = "shampoo_id",
referencedColumnName = "id"),
inverseJoinColumns = @JoinColumn(name = "ingredient_id",
referencedColumnName = "id"))
private Set<BasicIngredient> ingredients;
```

Shampoo implementations will store information about the ingredients they are made of. We will implement that by setting a bidirectional **Many-to-Many** relationship between the ingredients and each shampoo. A new annotation "**`JoinTable`**" is visible, which will create a mapping table in the database - "**shampoo_ingredients**". It will store the relation between each shampoo **`id`** and ingredient **`id`**.

The company produces 4 different types of Shampoos:

- **Fresh Nuke**
  - **Brand "Fresh Nuke"**
  - **Price 9.33**
  - **Size Big**
  - **It's made of Mint, Nettle and Ammonium Chloride**

- **Pink Panther**
  - **Brand "Pink Panther"**
  - **Price 8.50**
  - **Size Medium**
  - **It's made of Lavender and Nettle**
- **Fifty Shades**
  - **Brand "Fifty Shades"**
  - **Price 6.69**
  - **Size Small**
  - **It's made of Strawberry and Nettle**

FreshNuke:

```
@Entity
@DiscriminatorValue(value = "FN")
public class FreshNuke extends BasicShampoo {

    private static final String BRAND = "Fresh Nuke";

    private static final BigDecimal PRICE = new BigDecimal( val: "9.33");

    private static final Size SIZE = Size.LARGE;

    public FreshNuke() {
    }

    public FreshNuke(BasicLabel classicLabel) {
        super(BRAND, PRICE, SIZE, classicLabel);
    }
}
```

Create the other shampoo types analogically.
The inheritance should be presented in a single table.

Each shampoo can be 3 sizes: SMALL, MEDIUM or BIG. Create an enumeration which will hold the size information:

```
public enum Size {
    SMALL, MEDIUM, LARGE
}
```

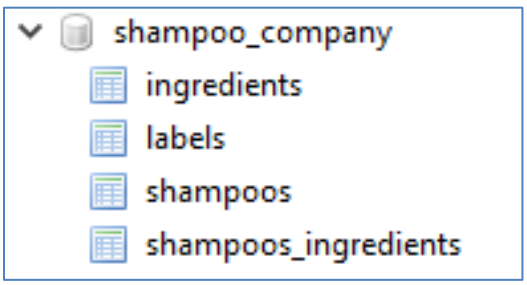
Make the field in the Shampoo class persistent by adding an **`@Enumerated`** annotation.

5. Test Application

Create a **FreshNuke** shampoo. Add **nettle**, **mint** and **ammonium chloride** to it's ingredients. Set up a label and persist the shampoo:

```
public class Main {
    public static void main(String[] args) {
        EntityManagerFactory managerFactory = Persistence
                .createEntityManagerFactory( persistenceUnitName: "shampoo_company");
        EntityManager em = managerFactory.createEntityManager();
        em.getTransaction().begin();

        BasicIngredient am = new AmmoniumChloride();
        BasicIngredient mint = new Mint();
        BasicIngredient nettle = new Nettle();

        BasicLabel label =
                new BasicLabel( title: "Fresh Nuke Shampoo",
                        subtitle: "Contains mint and nettle");

        BasicShampoo shampoo = new FreshNuke(label);

        shampoo.getIngredients().add(mint);
        shampoo.getIngredients().add(nettle);
        shampoo.getIngredients().add(am);
        em.persist(shampoo);

        em.getTransaction().commit();
        em.close();
    }
```

If you've implemented everything correctly, the following **database schema** will be created:



Ingredients table:

ingredient_type	id	name	price	chemical_formula
AM	1	Ammonium Chloride	6.12	NH4Cl
NT	2	Nettle	6.12	(NULL)
MN	3	Mint	3.54	(NULL)

Labels table:

id	subtitle	title
1	Contains mint and nettle	Fresh Nuke Shampoo

Shampoos table:

shampoo_type	id	brand	price	size	label
FN	0	Fresh Nuke	9.33	2	1

Shampoos and ingredients table(created via the **Many-toMany** relationship in the **Shampoo** class):

shampoo_id	ingredient_id
0	1
0	2
0	3