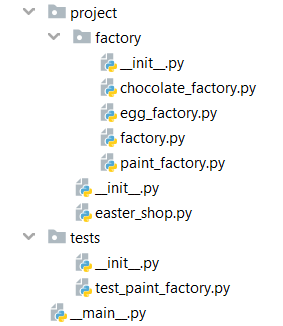
# Easter Shop Factories

*Easter is coming, so you are given a task to track all the products made in the chocolate, egg and paint factory, so you can sell them in your Easter shop and make people happy*

You will be provided with a **skeleton** which includes all the folders and files that you will need.

***Note: You are not allowed to change the folder and file structure and change their names!***



# Judge Upload

For the **first 2 problems**, create a **zip** file with the name **project** and upload it to the judge system

For the **last problem**, create a **zip** file with the name **tests** and upload it to the judge system

# Structure (Problem 1) and Functionality (Problem 2)

Our first task is to implement the **structure and functionality** of all the classes (attributes, methods, inheritance, etc.)

## Class Factory

In the file **factory.py** the class **Factory** should be implemented:

### Structure

The class should be abstract, and should have the following attributes:

* **name: str** - passed upon **initialization**
* **capacity: int** - passed upon **initialization**
* **ingredients: dict** - **empty** upon initialization (**name** of the ingredient for **key** and **quantity** of the ingredient as a **value**)

### Methods

#### \_\_init\_\_(name: str, capacity: int)

The **\_\_init\_\_** method should receive a **name: str**, and **capacity: int**. Set the attributes to the given ones

#### add\_ingredient (ingredient\_type: str, quantity: int)

Method should be **implemented** by the **child** classes

#### remove\_ingredient (ingredient\_type: str, quantity: int)

Method, should be **implemented** by the **child** classes

#### can\_add(value: int)

**returns** whether the given amount of product (value) **can be added** in the ingredients

## Class ChocolateFactory

In the **chocolate\_factory.py** file the class **ChocolateFactory** should be implemented

### Structure

The class should **inherit** from the **Factory** class and the following **additional attributes** should be added:

* **recipes: dict** - empty upon initialization (recipe name as key and dictionary of needed ingredients to make the recipe)
* **products: dict** - empty upon initialization (made recipes; recipe name as key and quantity as value)

### Methods

#### \_\_init\_\_(name: str, capacity: int)

Same as the **Factory** initialization but with the **recipes** and **products** attributes

#### add\_ingredient (ingredient\_type: str, quantity: int)

* If the **ingredient** type is **one of ["white chocolate", "dark chocolate", "milk chocolate", "sugar"]** and there is **enough space** in the factory, **add the ingredient** with its quantity to the **ingredients**. If the ingredient **already exists**, **increase** the quantity
* If there is **not enough space** - raise a **ValueError** with message **"Not enough space in factory"**
* If the ingredient is from the **wrong type**, raise a **TypeError** with message **"Ingredient of type {type} not allowed in {class name of the factory}"**

#### remove\_ingredient (ingredient\_type: str, quantity: int)

* If the **ingredient** type is in the **ingredients** and we have the given **quantity**, **remove** the given **quantity** of that ingredient
* If we **don't** have enough **quantity** - raise **ValueError** with message **"Ingredient quantity cannot be less than zero"**
* If we **don't** have the **ingredient** - raise **KeyError** with message **"No such product in the factory"**

#### add\_recipe(recipe\_name: str, recipe: dict)

* If the **recipe is new**, add it, otherwise **update** the recipe

#### make\_chocolate(recipe\_name: str)

* If we **have the recipe** in our recipes, **make** that **product (recipe name)** (add it to the products, or **increase** its quantity if we **already have** it in the products dictionary) and **remove the ingredients** we used from the ingredients dictionary
* If we don't have the recipe - raise **TypeError** with message **"No such recipe"**

## Class EggFactory

In the **egg\_factory.py** file the class **EggFactory** should be implemented

### Structure

The class should **inherit** from the **Factory** class

### Methods

#### \_\_init\_\_(name: str, capacity: int)

Same as the **Factory** initialization

#### add\_ingredient(ingredient\_type: str, quantity: int)

* If the ingredient type is **one of ["chicken egg", "quail egg"]** and there is **enough space** in the factory, **add the ingredient** with its quantity to the **ingredients**. If the ingredient **already exists**, **increase** the quantity
* If there is **not enough space** - raise a **ValueError** with message **"Not enough space in factory"**
* If the ingredient is from the **wrong type**, raise a **TypeError** with message **"** **Ingredient of type {type} not allowed in {class name of the factory}"**

#### remove\_ingredient(ingredient\_type: str, quantity: int)

* If the **ingredient type** is in the **ingredients** and we have the given **quantity**, **remove** the given **quantity** of that ingredient
* If we **don't** have enough **quantity** - raise **ValueError** with message **"Ingredient quantity cannot be less than zero"**
* If we **don't** have the **ingredient** - raise **KeyError** with message **"No such ingredient in the factory"**

### Properties

#### products()

* **products** should be a **property** that returns the ingredients

## Class PaintFactory

In the **paint\_factory.py** file the class **PaintFactory** should be implemented

### Structure

The class should **inherit** from the **Factory** class

### Methods

#### \_\_init\_\_(name: str, capacity: int)

Same as the **Factory** initialization

#### add\_ingredient(ingredient\_type: str, quantity: int)

* If the ingredient type is **one of ["white", "yellow", "blue", "green", "red"]** and there is **enough space** in the factory, **add the ingredient** with its quantity to the **ingredients**. If the ingredient **already exists**, **increase** the quantity
* If there is **not enough space** - raise a **ValueError** with message **"Not enough space in factory"**
* If the ingredient is from the **wrong type**, raise a **TypeError** with message **"Ingredient of type {type} not allowed in {class name of the factory}"**

#### remove\_ingredient(ingredient\_type: str, quantity: int)

* If the **ingredient type** is in the **ingredients** and we have the given **quantity**, **remove** the given **quantity** of that product
* If we **don't** have enough **quantity** - raise **ValueError** with message **"Ingredient quantity cannot be less than zero"**
* If we **don't** have the **ingredient** - raise **KeyError** with message **"No such ingredient in the factory"**

### Properties

#### products()

* **products** should be a **property** that returns the ingredients

## Class EasterShop

In the **easter\_shop.py** file the class **EasterShop** should be implemented. The **EasterShop** will be the class that handles all the factories

### Structure

The class should have the following attributes:

* **name: str** - passed upon initialization
* **chocolate\_factory: ChocolateFactory** - object passed upon initialization
* **egg\_factory: EggFactory** - object passed upon initialization
* **paint\_factory: PaintFactory** - object passed upon initialization
* **storage: dict** - empty upon initialization (product name as key and quantity of the product as value)

### Methods

#### add\_chocolate\_ingredient(type: str, quantity: int)

**adds** an ingredient in the **chocolate\_factory**

#### add\_egg\_ingredient(type: str, quantity: int)

**adds** an ingredient in the **egg\_factory**

#### add\_paint\_ingredient(type: str, quantity: int)

**adds** an ingredient in the **paint\_factory**

#### make\_chocolate(recipe: str)

**makes** a chocolate in the **chocolate\_factory** and add the **chocolate** to the **storage**

#### paint\_egg(color: str, egg\_type: str)

check the **egg factory** to see if you have at least **one egg** of the given egg type and check the **paint factory** if you have at least **one piece** of the given **color**

* If you have at **least one egg of the type** and **one color** of the **given color**, add a **painted egg** in the **storage** (key -> **"{color} {egg\_type}"**, value -> **1** if this is the **first** colored egg from the type, **increase** if the egg of that color **already exists**). You should also **remove** the **used ingredients** from both factories
* Otherwise raise **ValueError** with message **"Invalid commands"**

#### \_\_repr\_\_()

Returns a string representation of the shop in the following format:

**Shop name: {name}  
Shop Storage:  
{item1 name}: {item1 quantity}  
…**

# Problem 3. Unit Tests

Write tests for the **PaintFactory** in the **test\_paint\_factory** file