VE280 (23FA) Exercise 5

This exercise focuses on *Exception* and *ADT*. The deadline is **Nov 19, 2023, 23:59p.m.**

Description

A new vending machine is set on the 3rd floor of the Dragon Ice Building. Kurumi forgot all about eating and sleeping when struggling with her project, so she bought a bag of chips and two bars of chocolates from the new vending machine. However, she is still tired and would like to get some coffee to refresh herself. Considering that there is no coffee vendor in the Dragon Ice Building, Kurumi would like to design one on her own.

To implement her coffee vendor, she has fetched part of the architecture from the designer of the vending machine. She simplified some of the functions and reconstructed them into a `VendingMachine` class. For specific, there are two values in this object:

1. `price`: An array of integer representing the price of each type of food.
2. `type`: The most delicious food existing in the current vending machine.

In order to quantify how delicious the food is, she have conducted a survey among customers living in the Dragon Ice Building. She ranked all types of food according to the survey results and represented them as an `enum` type:

```
enum FoodType{
    Latte, // the least delicious
    Cappuccino,
    Americano,
    Milk,
    Chocolate,
    Bread,
    Chips,
    InstantNoodle,
    Biscuit,
    Sandwich, // the most delicious
};
```

She also found that Americano is the most popular type of coffee based on the survey, so she decided to maintain the existence of Americano in her coffee vendor.

There are several methods to facilitate the vending machine:

1. `setPrice`: Set the price of the given type of food. **If the price of the current food is 0, we say this type of food is "not existing".**
2. `getPrice`: Get the price of the given type of food.
3. `print`: Print the menu that lists all the existing food. **The listing order is from the most delicious one to the least delicious one.**
4. `getType`: Get the most delicious type of food existing.

Considering that the functions of the coffee vendor are similar to those of the vending machine, Kurumi decided to create a derived class `CoffeeVendor` based on the previous implementation. The difference is that a coffee vendor only sells coffee, which means that only **Latte, Cappuccino, and Americano** are available types in the current class. As a result, there are two methods to override:

1. `setPrice`: Set the price of the given type of coffee. **If the price of the current coffee is 0, we say this type of coffee is "not existing".**
2. `print`: Print the menu that lists all the existing coffee. **The listing order is from the most delicious one to the least delicious one.**

Since Kurumi hasn't completed her project yet, she would like to ask you for help.

Task

Here are some requirements that you should carefully read to design the vending machine and the coffee vendor.

- Exception

  Sometimes the machine might work incorrectly if the given type of food is not contained in the `FoodType` list or not exists in the current vending machine. To handle this issue, an exception class is provided:

  ```
  class Exception{

      std::string message="";

  public:
      Exception(const std::string & message): message(message) {}
      std::string what() {return message;}
  };
  ```

  You can throw an instance of `Exception` with a specific message using this class.

- Vending Machine

  The declaration of `VendingMachine` is provided in `vendingmachine.h`. You need to implement the `VendingMachine` class in your `vendingmachine.cpp`. Do not modify anything in `vendingmachine.h`.

  ```
  class VendingMachine {

  protected:
      // the price of each type of food
      int price[MAX_TYPE] = {0};
      // the most delicious type of food existing
      FoodType type;

  public:

      /**
       * @brief Default Constructer. Initialize all the price to zero,
       * and set this->type to be the least delicious food.
       */
  ```

```
    VendingMachine();

    /**
     * @brief Set the price of the food for the type given,
     * and update the most delicious type.
     * @param type
     * @param price
     * @throw If the given type is greater or equal to MAX_type, throw an
Exception
     * object with error message "Exceed maximum type!".
     */
    virtual void setPrice(FoodType type,int price);

    /**
     * @brief Get the price of the food for the type given.
     *
     * @param type
     * @throw If the given type is more delicious than this->type
     * , throw an Exception object with error message "No food of such
type!".
     */
    int getPrice(FoodType type) const;

    /**
     * @brief Print the menu with format "<food type> $<price>\n" to stdout,
     * from the most delicious type to least one, skip all the food with
price of zero.
     * Each line represents one type of food.
     * Add a new line after printing the whole menu.
     * @example Sandwich $12
     * @example Chocolate $5
     * @example Latte $1
     */
    virtual void print() const;

    /**
     * @return FoodType, the most delicious food existing
    */
    FoodType getType() const;

    ~VendingMachine() = default;

};
```

- Coffee Vendor

  The declaration of `CoffeeVendor` is provided in `vendingmachine.h`. You need to implement the `CoffeeVendor` class in your `vendingmachine.cpp`. Do not modify anything in `vendingmachine.h`.

```
class CoffeeVendor : public VendingMachine{

public:
    /**
     * @brief Default Constructer. Initialize the price of Americano to 1,
     * the price of Latte and Cappuccino to 0,
     * and set this->type to be Americano.
     */
    CoffeeVendor();

    /**
     * @brief Set the price of the coffee for the given type.
     * @param type
     * @param price
     * @throw If the given type is not a coffee type, throw an Exception
     * object with error message "Exceed maximum type!".
     * @throw If the given type is Americano and the given price is 0, throw
an Exception
     * object with error message "No Americano!".
     */
    void setPrice(FoodType type,int price);

    /**
     * @brief Print the menu with format "<coffee type> $<price>\n" to
stdout,
     * from the most delicious type to least one, skip all the coffee with
price of zero.
     * Each line represents one type of coffee.
     * Add a new line after printing the whole menu.
     * @example Americano $14
     * @example Cappuccino $10
     * @example Latte $6
     */
    void print() const;

    ~CoffeeVendor() = default;

};
```

Submission

Please compress your `vendingmachine.cpp` into `.zip` file and submit on JOJ.