

TAHIRI Abderrahmane

05 juillet 2023



OBJECTIVE DE MINI-PROJET

Le mini-projet que nous présentons est une application de simulation d'un distributeur automatique de snacks (snack vending machine) en C++. Cette application permet de gérer plusieurs emplacements (slots) contenant différents produits, tels que des chips, des bonbons et des sodas. L'objectif principal de cette simulation est de permettre aux utilisateurs d'acheter des produits en insérant la bonne quantité de pièces.

INTRODUCTION :

L'objectif de ce mini-projet est de créer une application de simulateur de distributeur automatique de snacks, qui permettra de comprendre le fonctionnement d'un tel système et d'expérimenter des interactions utilisateur avec celui-ci. Grâce à cette simulation, les utilisateurs pourront visualiser les produits disponibles, connaître leurs prix, vérifier les quantités restantes et effectuer des achats en insérant des pièces. Cela permettra de simuler le comportement d'un véritable distributeur automatique et d'offrir une expérience interactive réaliste. De plus, cette simulation peut servir de base pour développer des fonctionnalités supplémentaires, telles que la gestion des stocks, la prise en charge de différentes devises, etc.

DÉMARCHE :

Pour développer l'application de console en C++ pour simuler un distributeur automatique de snacks, vous pouvez suivre les étapes suivantes :

Créez les classes `Automat`, `Slot`, `Motor`, `CoinSlot`, `Keypad` et `DropCheck` en respectant les attributs et les méthodes décrits dans l'énoncé. Chaque classe doit être implémentée dans un fichier source (.cpp) et un fichier d'en-tête (.h) séparés.

Implémentez les constructeurs, destructeurs, getters et setters appropriés pour chaque classe.



Dans la classe `Automat`, implémentez les méthodes `addSlot()`, `searchSlot()`, `changeSlot()`, `getPieces()`, `isAvailable()`, `getPrice()`, `dropSlot()`, `fillAll()`, et `fill()` en fonction de leur description.

Dans la classe `Slot`, dérivez deux classes dérivées : `SmallSlot` et `WideSlot` en respectant la structure et les méthodes héritées de la classe de base `Slot`. Implémentez la méthode `drop()` dans les deux classes dérivées en fonction des spécifications de l'énoncé.


Dans la classe `CoinSlot`, implémentez les méthodes `updateCoinAmount()`, `getCoinAmount()`, `Clear()`, et `returnCoins()` en fonction de leur description.

Dans la classe `Keypad`, implémentez la méthode `getSelection()` pour demander à l'utilisateur de saisir le numéro de l'emplacement du produit.

Dans la classe `DropCheck`, implémentez la méthode `productReleased()` pour simuler la vérification de la chute du produit dans le bac de sortie.

Dans la classe `Motor`, implémentez la méthode `Trigger()` pour activer le moteur et afficher un message.

Dans la fonction principale de votre application, créez une instance de la classe `Automat` et utilisez une boucle de simulation pour gérer les différents états et actions du distributeur automatique en fonction des entrées de l'utilisateur.



Utilisez un commutateur dans la boucle de simulation pour exécuter les actions en fonction de l'état actuel du distributeur automatique. Suivez le diagramme d'état fourni dans l'énoncé pour déterminer les transitions d'état et les actions à effectuer dans chaque état.

Testez votre application en simulant différentes interactions de l'utilisateur avec le distributeur automatique, en vous assurant que les pièces sont correctement insérées, que les produits sont éjectés et que les pièces de monnaie sont rendues lorsque nécessaire.

Code source :

```
#include <iostream>
#include <vector>

// Définition de la classe moteur
class Motor {
private:
    int id;

public:
    Motor(int motorId) : id(motorId) {}

    void Trigger() {
        std::cout << "Motor " << id << " activated" << std::endl;
    }
};

// Définition de la classe Slot
```

```
class Slot {
private:
    std::string productName;
    int id;
    int numProducts;
    int price;
    Motor* motor1;
    Motor* motor2;

public:
    Slot(int slotId, std::string name, int productPrice, int motorId1, int
motorId2 = 0)
        : id(slotId), productName(name), numProducts(0),
price(productPrice), motor1(new Motor(motorId1)), motor2(motorId2 != 0 ?
new Motor(motorId2) : nullptr) {}

    ~Slot() {
        delete motor1;
        delete motor2;
    }

    void drop() {
        motor1->Trigger();
        if (motor2 != nullptr) {
            motor2->Trigger();
        }
        std::cout << productName << " delivered from slot " << id <<
std::endl;
    }

    // Getters and Setters
    std::string getProductName() const {
        return productName;
    }

    void setProductName(const std::string& name) {
        productName = name;
    }
}
```



```
    }

    int getNumProducts() const {
        return numProducts;
    }

    void setNumProducts(int num) {
        numProducts = num;
    }


    int getPrice() const {
        return price;
    }

    void setPrice(int productPrice) {
        price = productPrice;
    }

    int getId() const {
        return id;
    }
};

// Définition de la classe Automat
class Automat {
private:
    std::vector<Slot*> slots;
    int numSlots;
    int numProductsPerSlot;
    Slot* cached;

public:
    Automat(int numSlots, int maxProductsPerSlot)
        : numSlots(numSlots), numProductsPerSlot(maxProductsPerSlot),
        cached(nullptr) {
        slots.reserve(numSlots);
    }
};
```



```
~Automat() {
    for (Slot* slot : slots) {
        delete slot;
    }
}

void addSlot(Slot* slot) {
    slots.push_back(slot);
}

Slot* searchSlot(int slotId) {
    if (cached != nullptr && cached->getId() == slotId) {
        return cached;
    }

    for (Slot* slot : slots) {
        if (slot->getId() == slotId) {
            cached = slot;
            return slot;
        }
    }

    return nullptr;
}

void changeSlot(int slotId, const std::string& newName, int newPrice) {
    Slot* slot = searchSlot(slotId);
    if (slot != nullptr) {
        slot->setProductName(newName);
        slot->setPrice(newPrice);
    }
}

int getPieces(int slotId) {
    Slot* slot = searchSlot(slotId);
    return slot != nullptr ? slot->getNumProducts() : -1;
}
```



```
}

bool isAvailable(int slotId) {
    Slot* slot = searchSlot(slotId);
    return slot != nullptr && slot->getNumProducts() > 0;
}

int getPrice(int slotId) {
    Slot* slot = searchSlot(slotId);
    return slot != nullptr ? slot->getPrice() : -1;
}


void dropSlot(int slotId) {
    Slot* slot = searchSlot(slotId);
    if (slot != nullptr && slot->getNumProducts() > 0) {
        slot->drop();
        slot->setNumProducts(slot->getNumProducts() - 1);
    }
}

void fillAll(int numProducts) {
    for (Slot* slot : slots) {
        slot->setNumProducts(numProducts);
    }
}

void fill(int slotId, int numProducts) {
    Slot* slot = searchSlot(slotId);
    if (slot != nullptr) {
        slot->setNumProducts(numProducts);
    }
}

};

int main() {
    Automat vendingMachine(3, 10);
```



```
Slot* slot1 = new Slot(1, "Chips", 200, 1);
Slot* slot2 = new Slot(2, "Candy", 150, 2, 3);
Slot* slot3 = new Slot(3, "Soda", 300, 4);

vendingMachine.addSlot(slot1);
vendingMachine.addSlot(slot2);
vendingMachine.addSlot(slot3);

// Simulating user interactions
vendingMachine.fillAll(5);
std::cout << "Slot 1: " << vendingMachine.getPieces(1) << " pieces
available" << std::endl;
std::cout << "Slot 2: " << vendingMachine.getPieces(2) << " pieces
available" << std::endl;

if (vendingMachine.isAvailable(1)) {
    int price = vendingMachine.getPrice(1);
    std::cout << "Insert coins: " << price << " cents" << std::endl;
    vendingMachine.dropSlot(1);
} else {
    std::cout << "Product not available" << std::endl;
}

if (vendingMachine.isAvailable(2)) {
    int price = vendingMachine.getPrice(2);
    std::cout << "Insert coins: " << price << " cents" << std::endl;
    vendingMachine.dropSlot(2);
} else {
    std::cout << "Product not available" << std::endl;
}

delete slot1;
delete slot2;
delete slot3;

return 0;
}
```

