TP 11 : Gestion des Exceptions

Imad Kissami

21 Avril 2025

Objectif

- Créer un `Makefile` pour compiler tous les fichiers `.cpp`.
- Tous les fichiers doivent être regroupés dans un dossier `TP11_Nom_Prénom`.
- Mettre la main sur les concepts de gestion des exceptions.

Execice 1 :

Écrivez un programme qui demande à l'utilisateur de saisir deux nombres et une opération (+, -, *, /). Implémentez une fonction calculate qui effectue l'opération demandée et gère deux types d'exceptions :

- Une exception pour la division par zéro.
- Une exception pour une opération non valide (par exemple, si l'utilisateur entre un caractère autre que +, -, *, /).
- Utilisez une classe d'exception personnalisée pour chaque cas.

Instructions :

- Deux classes d'exceptions personnalisées (`DivideByZeroException` et `InvalidOperationException`) héritent de std ::exception et redéfinissent what() pour fournir des messages d'erreur clairs.
- La fonction calculate vérifie la validité de l'opération et lance les exceptions appropriées.
- Le bloc try-catch dans main gère les deux types d'exceptions spécifiques et inclut un catch-all (...) pour les erreurs inattendues.

Exemple du main.cpp :

```
int main() {
  double num1, num2;
  char operation;

  std::cout << "Enter␣first␣number:␣";
  std::cin >> num1;
  std::cout << "Enter␣second␣number:␣";
  std::cin >> num2;
  std::cout << "Enter␣operation␣(+,␣-,␣*,␣/):␣";
  std::cin >> operation;

  try {
    double result = calculate(num1, num2, operation);
    std::cout << "Result:␣" << result << std::endl;
  }
  catch (const DivideByZeroException& ex) {
    std::cerr << ex.what() << std::endl;
  }
  catch (const InvalidOperationException& ex) {
    std::cerr << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unknown␣error␣occurred" << std::endl;
  }

  std::cout << "Program␣completed" << std::endl;
  return 0;
}
```

Exercice 2 :

Créez une classe Inventory qui gère un stock d'articles (représenté par un entier). Implémentez des méthodes pour ajouter (`add_items`) et retirer (`remove_items`) des articles. Gerez les erreurs suivantes avec des exceptions :

- Tentative d'ajouter un nombre négatif d'articles.
- Tentative de retirer plus d'articles que le stock disponible. Utilisez des classes d'exceptions personnalisées et testez la classe dans un programme principal.

Instructions :

- Les classes `NegativeQuantityException` et `InsufficientStockException` sont similaires à `IllegalBalanceException` et InsufficientFundsException dans les exemples de comptes bancaires.
- La classe Inventory vérifie les conditions d'erreur dans le constructeur et les méthodes `add_items`/`remove_items`, lançant des exceptions si nécessaire.
- Le programme principal teste les cas d'erreur (quantité négative, retrait excessif) et gère les exceptions de manière spécifique.

Exemple du main.cpp :

```cpp
int main() {
  // Test 1: Initialisation et ajout d'articles
  Inventory inv(100); // Initial stock of 100 items
  std::cout << "Initial stock: " << inv.get_stock() << std::endl;

  try {
    inv.add_items(50);
    std::cout << "After adding 50 items: " << inv.get_stock() << std::endl;
  }
  catch (const NegativeQuantityException& ex) {
    std::cerr << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unknown error occurred" << std::endl;
  }

  // Test 2: Tentative de retrait excessif
  try {
    inv.remove_items(160); // Should throw InsufficientStockException
    std::cout << "After removing 160 items: " << inv.get_stock() << std::endl;
  }
  catch (const InsufficientStockException& ex) {
    std::cerr << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unknown error occurred" << std::endl;
  }

  // Test 3: Tentative d'ajout d'une quantité négative
  try {
    inv.add_items(-10); // Should throw NegativeQuantityException
    std::cout << "After adding -10 items: " << inv.get_stock() << std::endl;
  }
  catch (const NegativeQuantityException& ex) {
    std::cerr << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unknown error occurred" << std::endl;
  }

  std::cout << "Program completed" << std::endl;
  return 0;
}
```

Exercice 3 :

Implémentez une classe Stack qui gère une pile d'entiers avec une capacité maximale fixée au moment de la construction. Ajoutez des méthodes push et pop. Gerez les erreurs suivantes avec des exceptions :

— Tentative de pousser un élément dans une pile pleine.

— Tentative de retirer un élément d'une pile vide. Créez une série de fonctions (`func_a`, `func_b`, `func_c`) qui utilisent la pile et provoquent une exception pour démontrer le déroulement de pile (stack unwinding). Capturez l'exception dans main.

Instructions :

— La classe `Stack` utilise un std ::vector pour stocker les éléments et vérifie les conditions de pile pleine/vide dans push et pop.

— Les classes d'exceptions `StackFullException` et StackEmptyException sont similaires à `DivideByZeroException` et `NegativeValueException` dans les exemples précédents.
— Les fonctions `func_a`, `func_b`, `func_c` simulent une chaîne d'appels, comme dans l'exemple de déroulement de pile. L'exception est lancée dans `func_c` et propagée jusqu'à main.
— Le bloc `try-catch` dans main capture les exceptions spécifiques et affiche un message, démontrant le déroulement de pile.

Exemple du main.cpp :

```
int main() {
  // Test 1: Démontrer StackFullException via func_a
  try {
    Stack stack(2); // Stack with capacity of 2
    stack.push(1);
    stack.push(2);
    std::cout << "Stack␣size:␣" << stack.size() << std::endl;

    func_a(stack); // Will throw StackFullException
  }
  catch (const StackFullException& ex) {
    std::cerr << "Caught␣in␣main:␣" << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unknown␣error␣occurred" << std::endl;
  }

  // Test 2: Démontrer StackEmptyException avec pop
  try {
    Stack stack(2); // Nouvelle pile vide
    std::cout << "Stack␣size:␣" << stack.size() << std::endl;
    stack.pop(); // Will throw StackEmptyException
  }
  catch (const StackEmptyException& ex) {
    std::cerr << "Caught␣in␣main:␣" << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unknown␣error␣occurred" << std::endl;
  }

  std::cout << "Program␣completed" << std::endl;
  return 0;
}
```