Système de Gestion d'un Hôpital

Présentation du projet en C++

Réalisé par : AIT-IDIR Abdelkhalek
ADIDI Aymane

Introduction

- Dans le cadre de ce projet, nous avons développé un système de gestion d’hôpital en C++, conçu pour rationaliser et optimiser la gestion des prestations médicales.
- Ce système s’inscrit dans une démarche visant à améliorer l'efficacité administrative et à garantir une meilleure organisation des services hospitaliers.

Fonctionnalités principales

- Les principales fonctionnalités incluent la gestion des prestations suivantes :

  Consultations : Suivi des patients et des médecins spécialistes.

  Urgences : Gestion des interventions rapides et des traitements d’urgence.

  Chirurgies : Organisation des opérations avec suivi des chirurgiens et des salles.

  Analyses biologiques : Suivi des prélèvements et des examens de laboratoire.

  Radiologies : Gestion des imageries médicales avec suivi des doses et types d'examens.

Structure du Code

- Le système est organisé autour de plusieurs classes :
  - **Classe de base** : Prestation
  - **Classes dérivées** : Consultation, Urgence, Chirurgie, Analyse biologique, Radio
  - **Classe principale** : Hopital

- Chaque classe encapsule les données et méthodes nécessaires à la gestion des prestations.

Structure du Code

```cpp
// Classe de base
class Prestation {
private:
    string nom;
    string departement;
    int code;
public:
    Prestation(const string& nom, const string& departement, int code)
    : nom(nom), departement(departement), code(code) {}
    virtual ~Prestation() = default;

    virtual void afficher() const;
    virtual void saisir();

    int getCode() const { return code; }
    const string& getNom() const { return nom; }

    bool operator==(const Prestation& other) const { return code == other.code; }
    friend bool operator<(const Prestation& lhs, const Prestation& rhs) { return lhs.nom < rhs.nom; }
};
```

- **Classe de base** : Prestation

```cpp
// Méthode de la classe de base Prestation
void Prestation::afficher() const {
    cout << "Nom: " << nom << ", Département: " << departement << ", Code: " << code << endl;
}

void Prestation::saisir() {
    cout << "Saisir le nom: ";
    cin.ignore();
    getline(cin, nom);
    cout << "Saisir le département: ";
    getline(cin, departement);
    cout << "Saisir le code: ";
    cin >> code;
}
```

```cpp
// Méthode de la classe dérivée 1
void PrestationConsultation::afficher() const {
    Prestation::afficher();
    cout << "Médecin: " << nomMedecin << ", Spécialité: " << specialite << endl;
}

void PrestationConsultation::saisir() {
    Prestation::saisir();
    cout << "Saisir le nom du médecin: ";
    cin.ignore();
    getline(cin, nomMedecin);
    cout << "Saisir la spécialité: ";
    getline(cin, specialite);
}
```

```cpp
// Classe dérivée 1
class PrestationConsultation : public Prestation {
    string nomMedecin;
    string specialite;

public:
    PrestationConsultation(const string& nom, const string& departement, int code, const string& nomMedecin, const string& specialite)
    : Prestation(nom, departement, code), nomMedecin(nomMedecin), specialite(specialite) {}
    void afficher() const override;
    void saisir() override;
};
```

Structure du Code

- **Classes dérivées** : Consultation

```cpp
// Classe dérivée 2
class PrestationUrgence : public Prestation {
    string typeUrgence;
    string medicamentsInjectes;

public:
    PrestationUrgence(const string& nom, const string& departement, int code, const string& typeUrgence, const string& medicamentsInjectes)
    : Prestation(nom, departement, code), typeUrgence(typeUrgence), medicamentsInjectes(medicamentsInjectes) {}
    void afficher() const override;
    void saisir() override;
};
```

```cpp
// Méthode de la classe dérivée 2
void PrestationUrgence::afficher() const {
    Prestation::afficher();
    cout << "Type d'urgence: " << typeUrgence << ", Médicaments injectés: " << medicamentsInjectes << endl;
}

void PrestationUrgence::saisir() {
    Prestation::saisir();
    cout << "Saisir le type d'urgence: ";
    cin.ignore();
    getline(cin, typeUrgence);
    cout << "Saisir les médicaments injectés: ";
    getline(cin, medicamentsInjectes);
}
```

Structure du Code

Classes dérivées : Urgence

```cpp
// Classe dérivée 3
class PrestationChirurgie : public Prestation {
    string nomChirurgien;
    int numeroSalle;

public:
    PrestationChirurgie(const string& nom, const string& departement, int code, const string& nomChirurgien, int numeroSalle)
    : Prestation(nom, departement, code), nomChirurgien(nomChirurgien), numeroSalle(numeroSalle) {}
    void afficher() const override;
    void saisir() override;
};
```

```cpp
// Méthode de la classe dérivée 3
void PrestationChirurgie::afficher() const {
    Prestation::afficher();
    cout << "Chirurgien: " << nomChirurgien << ", Salle: " << numeroSalle << endl;
}

void PrestationChirurgie::saisir() {
    Prestation::saisir();
    cout << "Saisir le nom du chirurgien: ";
    cin.ignore();
    getline(cin, nomChirurgien);
    cout << "Saisir le numéro de la salle: ";
    cin >> numeroSalle;
}
```

Structure du Code

Classes dérivées : Chirurgie

```cpp
// Méthode de la classe dérivée 4
void PrestationAnalyse::afficher() const {
    Prestation::afficher();
    cout << "Quantité de sang: " << quantiteSang << ", Type d'analyse: " << typeAnalyse << endl;
}

void PrestationAnalyse::saisir() {
    Prestation::saisir();
    cout << "Saisir la quantité de sang (en ml): ";
    cin >> quantiteSang;
    cout << "Saisir le type d'analyse: ";
    cin.ignore();
    getline(cin, typeAnalyse);
}
```

```cpp
// Classe dérivée 4
class PrestationAnalyse : public Prestation {
    int quantiteSang;
    string typeAnalyse;

public:
    PrestationAnalyse(const string& nom, const string& departement, int code, int quantiteSang, const string& typeAnalyse)
    : Prestation(nom, departement, code), quantiteSang(quantiteSang), typeAnalyse(typeAnalyse) {}
    void afficher() const override;
    void saisir() override;
};
```

Structure du Code

Classes dérivées : Analyse biologique

```cpp
// Méthode de la classe dérivée 5
void PrestationRadio::afficher() const {
    Prestation::afficher();
    cout << "Dosage de radiation: " << dosageRadiation << ", Type de radio: " << typeRadio << endl;
}

void PrestationRadio::saisir() {
    Prestation::saisir();
    cout << "Saisir le dosage de radiation: ";
    cin >> dosageRadiation;
    cout << "Saisir le type de radio: ";
    cin.ignore();
    getline(cin, typeRadio);
}
```

```cpp
// Classe dérivée 5
class PrestationRadio : public Prestation {
    double dosageRadiation;
    string typeRadio;

public:
    PrestationRadio(const string& nom, const string& departement, int code, double dosageRadiation, const string& typeRadio)
    : Prestation(nom, departement, code), dosageRadiation(dosageRadiation), typeRadio(typeRadio) {}
    void afficher() const override;
    void saisir() override;
};
```

Structure du Code

Classes dérivées : Radio

Structure du Code

- **Classe principale** : Hopital

```
// Classe principale
class Hopital {
    vector<Prestation*> prestations;

public:
    ~Hopital();
    void ajouterPrestation(Prestation* prestation);
    void supprimerPrestation(int code);
    void afficherPrestations() const;
    void trierPrestations();
    void sauvegarderDansFichier(const string& nomFichier) const;
    void reinitialiserBase();
    void afficherStatistiques() const;
};
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

  1. Destructeur
  2. L’ ajout d’une prestation

```
// Méthodes de la classe principale Hopital
Hopital::~Hopital() {
    for (Prestation* prestation : prestations)
        delete prestation;
}

void Hopital::ajouterPrestation(Prestation* prestation) {
    prestations.push_back(prestation);
}
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

    3. Suppression d’une prestation

```
void Hopital::supprimerPrestation(int code) {
    for (auto it = prestations.begin(); it != prestations.end(); ++it) {
        if ((*it)->getCode() == code) {
            delete *it;
            prestations.erase(it);
            cout << "Prestation supprimée avec succès.\n";
            return;
        }
    }
    cout << "Code non trouvé.\n";
}
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

    4. Affichage des prestations enregistrées

```
void Hopital::afficherPrestations() const {
    if (prestations.empty()) {
        cout << "Aucune prestation enregistrée.\n";
        return;
    }
    for (const Prestation* prestation : prestations) {
        prestation->afficher();
        cout << "-----------------\n";
    }
}
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

    5. Tri des prestations par nom

```
void Hopital::trierPrestations() {
    sort(prestations.begin(), prestations.end(), [](Prestation* a, Prestation* b) {
        return *a < *b;
    });
    cout << "Prestations triées avec succès.\n";
}
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

    6. Sauvegarde des données dans un fichier

```
void Hopital::sauvegarderDansFichier(const string& nomFichier) const {
    ofstream fichier(nomFichier);
    if (!fichier) {
        cout << "Erreur lors de l'ouverture du fichier.\n";
        return;
    }
    for (const Prestation* prestation : prestations) {
        fichier << prestation->getNom() << "\n";
    }
    cout << "Données sauvegardées dans le fichier " << nomFichier << " avec succès.\n";
}
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

    7. Réinitialisation de la base de données

```
void Hopital::reinitialiserBase() {
    for (Prestation* prestation : prestations)
        delete prestation;
    prestations.clear();
    cout << "Base de données réinitialisée.\n";
}
```

Fonctionnalités

- Les fonctionnalités principales du système incluent :

    8. Affichage des statistiques sur les prestations

```
void Hopital::afficherStatistiques() const {
    int consultations = 0, urgences = 0, chirurgies = 0, analyses = 0, radios = 0;
    for (const Prestation* prestation : prestations) {
        if (dynamic_cast<const PrestationConsultation*>(prestation)) ++consultations;
        else if (dynamic_cast<const PrestationUrgence*>(prestation)) ++urgences;
        else if (dynamic_cast<const PrestationChirurgie*>(prestation)) ++chirurgies;
        else if (dynamic_cast<const PrestationAnalyse*>(prestation)) ++analyses;
        else if (dynamic_cast<const PrestationRadio*>(prestation)) ++radios;
    }
    cout << "Statistiques:\n";
    cout << "Consultations: " << consultations << "\n";
    cout << "Urgences: " << urgences << "\n";
    cout << "Chirurgies: " << chirurgies << "\n";
    cout << "Analyses: " << analyses << "\n";
    cout << "Radios: " << radios << "\n";
}
```

La Fonction main

Cette fonction **main** orchestre l'interaction entre l'utilisateur et le système à travers un menu interactif.

1. **Affichage du Menu Principal**
   La fonction utilise la méthode **afficherMenu** pour présenter un menu clair et intuitif, offrant plusieurs choix à l'utilisateur, notamment :

```
void afficherMenu() {
    cout << "\n===== MENU =====\n";
    cout << "1. Ajouter une prestation\n";
    cout << "2. Afficher toutes les prestations\n";
    cout << "3. Supprimer une prestation\n";
    cout << "4. Trier les prestations\n";
    cout << "5. Sauvegarder les prestations dans un fichier\n";
    cout << "6. Réinitialiser la base\n";
    cout << "7. Afficher les statistiques\n";
    cout << "8. Quitter\n";
    cout << "Votre choix : ";
}
```

La Fonction main

2. **Gestion des Choix**
   Le menu repose sur une boucle **do-while**, qui reste active jusqu'à ce quel'utilisateur décide de quitter le programme. Chaque choix est traité dans un bloc **switch**, avec des appels à des méthodes spécifiques de la classe **Hopital** :

- **Par exemple :**

  le choix d'ajouter une prestation appelle la méthode **creerPrestation**, qui retourne une instance appropriée (comme **PrestationConsultation** ou **PrestationRadio**), ajoutée ensuite au système via **ajouterPrestation**.

3. **Gestion des Erreurs d'Entrée**

   La fonction inclut une gestion des erreurs d'entrée utilisateur grâce aux méthodes **cin.clear** et **cin.ignore**, garantissant la robustesse et la fluidité du programme.

Exemple d’Utilisation

1. Ajouter une prestation de type Consultation :

- Nom : Consultation générale
- Département : Médecine
- Code : 101
- Médecin : Dr. Dupont
- Spécialité : Généraliste

2. Afficher toutes les prestations pour vérifier l’ajout.

```
===== MENU =====
1. Ajouter une prestation
2. Afficher toutes les prestations
3. Supprimer une prestation
4. Trier les prestations
5. Sauvegarder les prestations dans un fichier
6. Réinitialiser la base
7. Afficher les statistiques
8. Quitter
Votre choix : 1

Type de prestation à ajouter :
1. Consultation
2. Urgence
3. Chirurgie
4. Analyse biologique
5. Radio
Votre choix : 1
Nom de la prestation : Consultation générale
Département : Médecine
Code : 101
Nom du médecin : DR. Dupont
Spécialité : Généraliste
Prestation ajoutée avec succès !
```

```
===== MENU =====
1. Ajouter une prestation
2. Afficher toutes les prestations
3. Supprimer une prestation
4. Trier les prestations
5. Sauvegarder les prestations dans un fichier
6. Réinitialiser la base
7. Afficher les statistiques
8. Quitter
Votre choix : 2

Liste des prestations :
Nom: Consultation gnrale, Département: Mdecine, Code: 101
Médecin: DR. Dupont, Spécialité: Gnraliste
```

Conclusion

Ce projet illustre l'utilisation de la programmation orientée objet en C++ pour résoudre un problème pratique.

Points forts :

- Modularité grâce à l'héritage.
- Gestion efficace des données avec des conteneurs STL.

Améliorations possibles :

- Ajout d'une interface graphique.
- Intégration avec une base de données externe.