Inheritance

Inheritance is used to derive new classes from existing ones, inheriting their properties and behaviors.

class MedicalRecord : public Patient

MedicalRecord inherits from Patient. It has access to the protected and public members of Patient.

class Doctor : public MedicalRecord

Doctor inherits from MedicalRecord. It has access to the protected and public members of both MedicalRecord and Patient.

class Appointment : public Doctor

Appointment inherits from Doctor. It has access to the protected and public members of Doctor, MedicalRecord, and Patient.

Encapsulation

Encapsulation is the concept of bundling data and methods that operate on that data within one unit, typically a class, and restricting direct access to some of the object's components.

- **Private members:**
- a_id, location, and reason in Appointment are private members, meaning they cannot be accessed directly outside the class.
- **Protected members:**
- id, name, and age in Patient are protected members, meaning they can be accessed within the class itself and by derived classes.
- patientid, treatment, and test in MedicalRecord are also protected.
- **Public methods:**
- Methods such as setId(int id), setAge(int age), show(), input(), display(), Appoint(int x, string y, string z), and print() provide controlled access to the class members.

Exception Handling

Exception handling is used to manage errors and exceptional situations.

- The setId and setAge methods in the Patient class throw invalid_argument exceptions if the provided values are invalid.

```
void setId(int id) {
    if (id <= 0) throw invalid_argument("Invalid ID: ID must be greater than 0");
    this->id = id;
}

void setAge(int age) {
    if (age <= 20) throw invalid_argument("Invalid Age: Age must be greater than 20");
    this->age = age;
}
```

- Exception handling in main():

```
while (true) {
    try {
        int userId;
        int userAge;
        cout << "Enter a valid user age: ";
        cin >> userAge;
        cout << "Enter a valid patient ID: ";
        cin >> userId;
        aa.setId(userId);
        aa.setAge(userAge);
        break;  // Exit loop if no exception is thrown
    } catch (const invalid_argument &e) {
        cerr << "Error: " << e.what() << ". Please try again." << endl;
    }
}
```

Polymorphism

Polymorphism is the ability to call the same method on different objects and have each of them respond in their own way. In this code, polymorphism is demonstrated through method overriding.

- The show() method in MedicalRecord overrides the show() method in Patient

```
void show() {
   cout << "Medical Record of Patient id is: " << patientid << endl;
   cout << "Treatment of Patient is: " << treatment << endl;
   cout << "Test of Patient is: " << test << endl;
   Patient::show();
}
```

Conclusion

- Inheritance: MedicalRecord inherits from Patient, Doctor inherits from MedicalRecord, and Appointment inherits from Doctor.
- Encapsulation: Data members are private or protected, with public methods providing access and modification functionality.
- Exception Handling: invalid_argument exceptions are thrown and caught to handle invalid input.
- Polymorphism: The show() method is overridden in the MedicalRecord class to extend its functionality while still calling the base class show() method.

Repository Link

https://github.com/momii69/OOP.git