#include "stdafx.h"

#include "ErrorCheck.h"

#include "Accounts.h"

#include <iostream>

#include <string>

#include <array>

#include <vector>

#include <conio.h>

#include <ctime>

using namespace std;

// START OF ACCOUNT BEHAVIOURS

// START OF ID BEHAVIOURS

// Generates a new ID based on the previous objects id incremented by 1

void ID::GenerateID(\_\_int16 iPreviousIDPass)

{

iID = ++iPreviousIDPass; // Increment ID by 1

}

// Outputs object ID

void ID::OutputID()

{

cout << iID << '\n';

}

// Returns object ID for future objects to generate a new one off the previous

\_\_int16 ID::ReturnID()

{

return iID;

}

// END OF ID BEHAVIOURS

// START OF OBJECTNUMBER BEHAVIOURS

// Gets number of objects for object creation

\_\_int16 ObjectNumber::GetNumberOfObjects(const string &sObjectTypePass)

{

\_\_int16 iNumber = 0;

// If the object is of type Doctor

if (sObjectTypePass == "doctor")

{

cout << "Please enter the number of " << sObjectTypePass << " accounts you would like to create (minimum of 2)\n";

}

// For other objects

else

{

cout << "Please enter the number of " << sObjectTypePass << " accounts you would like to create\n";

}

cin >> iNumber;

// Checks if bad input has been entered such as letters or special characters

iNumber = OnlyIntegerErrorChecking(iNumber);

// Checks if an integer is 0 or less (negative numbers)

iNumber = ZeroOrHigherErrorChecking(sObjectTypePass, iNumber);

cout << '\n';

return iNumber;

}

// END OF OBJECTNUMBER BEHAVIOURS

// START OF PERSON BEHAVIOURS

// Gets account username and password details

void Person::GetDetails(\_\_int16 &iNumberPass)

{

cout << "Please enter a username for account " << iNumberPass + 1 << "\n";

cin >> sName;

cout << "Please enter a password for account " << iNumberPass + 1 << "\n";

sPassword = HidePasswordDetails();

}

// Displays login details for test purposes

void Person::DisplayDetails(\_\_int16 &iNumberPass)

{

cout << iNumberPass + 1 << ": Name: " << sName << ", Password: " << sPassword << "\n\n";

}

// Checks username and password details to see if they are correct or incorrect

bool Person::Login(const string &sNamePass, const string &sPasswordPass)

{

bool bCheck;

// If the details are correct

if (sNamePass == sName && sPasswordPass == sPassword)

{

bCheck = true;

}

// If the details are incorrect

else

{

bCheck = false;

}

return bCheck;

}

// Displays login confirmation

void Person::LoginConfirmation(bool &bCheckPass)

{

if (bCheckPass == true)

{

cout << "Login details correct\n";

}

else

{

cout << "You have been Locked Out for incorrectly entering login details multiple times\n";

}

}

// Displays incorrect details message

void Person::IncorrectDetails()

{

cout << "Incorrect Details entered\n";

}

// Returns name back to main for booking

string Person::ReturnName()

{

return sName;

}

// END OF PERSON BEHAVIOURS

// START OF DOCTOR BEHAVIOURS

// Doctor constructor

Doctor::Doctor()

{

iID = 0; // Doctor starting ID

iAppointmentSlots = 10; // Assigns all instances of doctor to 10 available slots

}

// Doctor destructor

Doctor::~Doctor()

{

}

// Gets specialist area information

void Doctor::GetDoctorDetails(\_\_int16 &iNumberPass)

{

cout << "\nPlease enter the doctor's specialist area for doctor" << iNumberPass + 1 << "\n";

cin >> sSpecialistArea;

cout << '\n';

}

// Show doctor details for booking options

void Doctor::ShowDoctorDetails(\_\_int16 &iNumberPass)

{

cout << iNumberPass + 1 << ": Name: " << sName << ", Specialist Area: " << sSpecialistArea << ", Available Slots: " << iAppointmentSlots << '\n';

}

// Removes appointment slot for doctor when a booking is created with the doctor

void Doctor::MinusAppointmentSlot()

{

iAppointmentSlots = --iAppointmentSlots;

}

// Adds back appointment slot for doctor when booking is completed or removed

void Doctor::AddAppointmentSlot()

{

iAppointmentSlots = ++iAppointmentSlots;

}

// Checks if the doctor currently has any available slots

bool& Doctor::NoAvailableSlots(bool &bCheckPass)

{

bCheckPass = false;

// If there are no slots left for the doctor, deny the user from selecting that doctor choice

if (iAppointmentSlots == 0)

{

bCheckPass = false;

cout << "This doctor currently has no available slots, please choose a different doctor\n";

}

// Else there are available slots for the doctor, allow the user to select that doctor choice

else

{

bCheckPass = true;

}

return bCheckPass;

}

// END OF DOCTOR BEHAVIOURS

// START OF SURGERY BEHAVIOURS

// Surgery constructor

Surgery::Surgery()

{

iAvailableSlots = 10; // Assigns all surgery objects to have 10 available slots

}

// Surgery destructor

Surgery::~Surgery()

{

}

// Gets name, location, doctor and available appointment slot information

void Surgery::GetSurgeryDetails(\_\_int16 &iNumberPass)

{

cout << "Please enter the name for surgery " << iNumberPass + 1 << " (do not type 'surgery' at the end)\n";

cin >> sSurgeryName;

sSurgeryName = sSurgeryName + " surgery"; // Adds 'surgery' to the type inputted by the user to keep data consistent

cout << "Please enter the location for surgery " << iNumberPass + 1 << "\n";

cin >> sLocation;

cout << "Please enter the name of the doctor who does " << sSurgeryName << " for surgery " << iNumberPass + 1 << "\n";

cin >> sDoctor;

cout << '\n';

}

// Show surgery name, location, doctor and available slot information for booking

void Surgery::ShowSurgeryDetails(\_\_int16 &iNumberPass)

{

cout << iNumberPass + 1 << ": Surgery Type: " << sSurgeryName << ", Location: " << sLocation << ", Doctor: " << sDoctor

<< ", Available Slots: " << iAvailableSlots << '\n';

}

// Show surgery details for specific booked customer appointment

void Surgery::ShowSurgeryBookedDetails()

{

cout << "Surgery: " << sSurgeryName << ", Location: " << sLocation << "\n\n";

}

// Removes available slot for surgery when a booking is created

void Surgery::MinusAvailableSlot()

{

iAvailableSlots = --iAvailableSlots;

}

// Adds available slot for surgery when a booking is deleted

bool& Surgery::NoAvailableSlots(bool &bCheckPass)

{

bCheckPass = false;

if (iAvailableSlots == 0)

{

bCheckPass = false; // Deny the user from selecting surgery choice

cout << "This surgery currently has no available slots, please choose a different surgery\n\n";

}

else

{

bCheckPass = true;

}

return bCheckPass;

}

// Adds back available slot when booking tied to surgery has been deleted/completed

void Surgery::AddAvailableSlot()

{

iAvailableSlots = ++iAvailableSlots;

}

// END OF SURGERY BEHAVIOURS

// START OF CUSTOMER BEHAVIOURS

// Customer destructor

Customer::~Customer()

{

}

// Customer constructor

Customer::Customer()

{

iID = 0; // Assigns ID of 0 to the first customer object

}

// Get customer phone number, address and ailment details

void Customer::GetCustomerDetails(\_\_int16 &iPreviousIDPass)

{

cout << "Please enter the phone number for customer " << iPreviousIDPass + 1 << "\n";

cin >> iPhoneNumber;

iPhoneNumber = PhoneNumberOnlyIntegerErrorChecking(iPhoneNumber); // Error checking for only integer input

cout << "Please enter the address for customer " << iPreviousIDPass + 1 << "\n";

cin.ignore(); // Clears previous cin input to allow getline input

getline(cin, sAddress); // Gets all letters, white space and words on 1 line and stores

cout << '\n';

cout << "Please enter the ailment for customer " << iPreviousIDPass + 1 << "\n";

getline(cin, sAilment); // Gets all letters, white space and words on 1 line and stores

cout << '\n';

}

// Shows ailment of the customer in booking section

void Customer::ShowAilment(\_\_int16 &iPreviousIDPass)

{

cout << "\nCustomer " << iPreviousIDPass + 1 << "'s ailment: " << sAilment << '\n';

}

// Returns customer ailment back to main for booking details

string Customer::ReturnAilment()

{

return sAilment;

}

// Displays all customer details

void Customer::ShowCustomerDetails(\_\_int16 &iNumberPass)

{

cout << iNumberPass + 1 << ": Name: " << sName << ", Password: " << sPassword << ", PhoneNumber: " << iPhoneNumber << ", Address: " << sAddress << ", Ailment: " << sAilment << '\n';

}

// END OF CUSTOMER BEHAVIOURS

// END OF ACCOUNT BEHAVIOURS