

OOP Programming Samantha John 2115880

by Samantha John

Submission date: 10-Dec-2021 01:57PM (UTC+0000)

Submission ID: 165768604

File name:

33192_Samantha_Jay_John_OOP_Programming_Samantha_John_2115880_382345_1730714768.docx
(214.88K)

Word count: 5673

Character count: 29279



ADVANCED SOFTWARE DEVELOPMENT

Objected Oriented Programming



NOVEMBER 19, 2021
SAMANTHA JOHN

Contents

Introduction	2
Importance of Object-Oriented Programming	3
Key Concepts of Object-Oriented Programming	4
Class	4
Objects	5
Encapsulation	5
Inheritance	6
Polymorphism	7
Abstraction	8
Conclusion	9
References	9
Testing	0
Appendix	0
Video Link	0
Code	0

Introduction

This report will examine the importance of Object-Oriented Programming and discuss the key concepts with particular relation to a basic two player digital game such as tic-tac-toe. This is a simple two player game that is based around a 3 x 3 grid, players take turns to place their counter in a grid space trying to get three counters in a row. Through the use of OOP this game will demonstrate the use of classes, objects, attributes and methods as well as many more key concepts.

Sectional Word Count (87)

Importance of Object-Oriented Programming

Object Oriented Programming allows us to break a programme down into smaller problems, each problem can then be solved in turn and later incorporated with the other units of code to form a functioning programme. One advantage of OOP is that it enables us to solve problems in a more natural way through the bottom-up approach rather than the top-down approach, that languages such as C and COBAL use.

Sectional Word Count (72)

Key Concepts of Object-Oriented Programming

Class

A class, is a user defined datatype that has two key aspects. One being the data members which can be thought of as variables of the class which are used to define the properties. The other being the member functions which are the methods used to manipulate the data members within the class. Both types of members can be accessed by creating an instance of the class. In essence a class acts as the template for an object.

In the project Game an example of a class would be the Menu class, this represents the real-world object of a user selection menu and game interface that allows the user to select from a number of pre-defined choices. Methods such as `DisplayMenu()` ;and `DisplayRules()` allow the user to navigate around the menu and sub menu, whereas `turn()`, `win()` and `gameOver()` methods allow the players to engage in a simple game of tic-tac-toe.

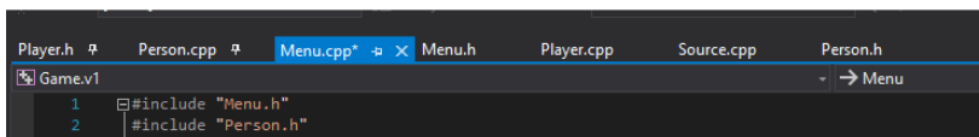


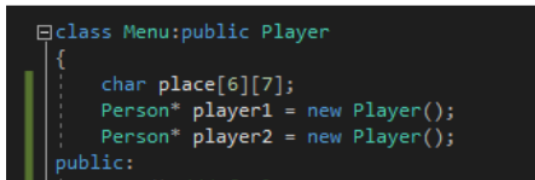
Figure 1 screen snip to show the varying classes within the project Game

Objects

An object is an instance of a class. When an object is created it is allocated memory within the programme. By creating an object of a class this allows us to access and make use of the methods in the class.

An example of an object in the Game project would be the creation of person1, the following line of code was used to create and test a Person object called person1: `Person* person1 = new Person();`

This line of code was inserted into the source.cpp file of the project in order to test that the object was created and initialised correctly. This line of code was later inserted into the menu.h file in order to create two player objects which represent the two players that will play a game of tic-tac-toe against each other.



```
class Menu:public Player
{
    char place[6][7];
    Person* player1 = new Player();
    Person* player2 = new Player();
public:
```

Figure 2 screenshot of menu.h file to show the creation of two player objects

Encapsulation

Encapsulation is the binding together of the data and related functions that are used to manipulate the data it also ensures that similar code and data is kept in the same place.



An example of this in the Game project would be in the menu class, here the variables (data members) row and column which are used in the function (member function) turn to check if the move the player is making is firstly valid and secondly if the player achieves three counters in a row.

Inheritance

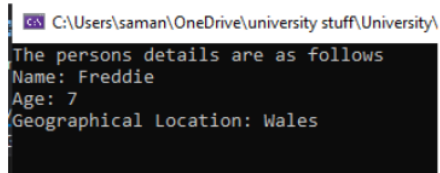
Inheritance allows classes to inherit properties or characteristics of other classes, which like polymorphism allows us to reuse code and lends to a more efficient way of programming.

With the use of inheritance, classes can be created to hold common attributes that can then be used across multiple classes.

In Object Oriented Programming a base class also known as a parent class would contain more generalised or common attributes that represent the object in question. Whereas the derived class often referred to as the child class can make use of these generalised attributes through the use of inheritance and it can specific it's own unique methods unique.

An example of inheritance in the Game project would be the classes Person and Player. In this example the Person class is the parent class, and the Player class is the child class. This is based on the real-world situation that every player is a person and both a person and player will have a name, age and geographical location but only a player will have a game score and game scoreboard position.

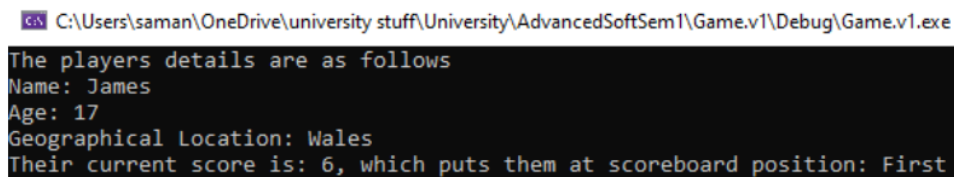
The first example shows the creation a Person object, this person has a name 'Freddie' who is 7 years old and lives in Wales.



```
C:\Users\saman\OneDrive\university stuff\University\AdvancedSoftSem1\Game.v1\Debug\Game.v1.exe
The persons details are as follows
Name: Freddie
Age: 7
Geographical Location: Wales
```

Figure 3 shows the code execution for a person object

Whereas this example shows the creation of a Player object, that also can be thought of as person that has a name 'James', who is 17 years old and also lives in Wales but we can also see that James is of type player and therefore has a score of 6 and is currently in first place.



```
C:\Users\saman\OneDrive\university stuff\University\AdvancedSoftSem1\Game.v1\Debug\Game.v1.exe
The players details are as follows
Name: James
Age: 17
Geographical Location: Wales
Their current score is: 6, which puts them at scoreboard position: First
```

Figure 4 shows the code execution for a player object

Polymorphism

According to W3schools polymorphism means many forms [1]. In Object Oriented Programming this is achieved when multiple classes are linked together through the use of inheritance. Classes that inherit from each other can make use of the same function and create new object of the base class.

Furthermore, through the use of polymorphism objects can respond in unique ways to the same message, this allows for code reuse and making for a more efficient way of programming,

From the project Game an example of polymorphism would be the creation of a parent class Person and the child class Player, in the real world a player is a person. In this example a person would have a name, age and geographical location as would a player, but a player would also have a score and scoreboard position within the game. In both classes the method printPerson() is executable but it achieves slightly different results in each class.  2

In the Person class if the printPerson() method is called it would only print out the name, age and geographical location assigned to the object but if the printPerson() method is called on a Player object this would print the name, age, geographical location as well as the score and scoreboard position of the object. With the use of polymorphism, it is possible to create a Player object of type person, but the compiler will still know that the object is of type Player and use the player printPerson() method instead.

```
The persons details are as follows
Name: Freddie
Age: 7
Geographical Location: Wales

The players details are as follows
Name: James
Age: 17
Geographical Location: Wales
Their current score is: 6, which puts them at scoreboard position: First
```

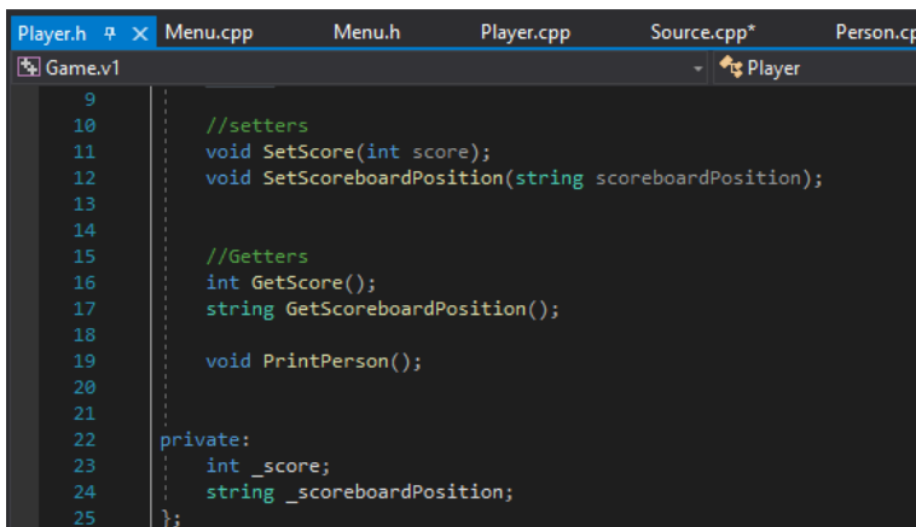
Figure 5 screenshot to show the successful execution of the virtual method

Further details of this test can be found in the [sample test](#) – Test number 3

Abstraction

James Hartman [2] suggests that abstraction is one of the most important concepts in Object Oriented Programming. He also explained that the main purpose of abstraction is to only show the essential attributes, hiding any unnecessary details from the user. The users can then access the attributes through the use of public setter and getter methods. Therefore, it would be fair to say that abstraction helps developers to reduce the complexity of the programme, making it easier for the user.  3

An example of abstraction in the Game project is the use of the private access modified in the Player class, here the attributes `_score` and `_scoreboardPosition` are declared as private. In order for the user to access these attributes a series of setter and getter methods would need to be created, for example `getScore()` and `setScoreboardPosition()`.



```
Player.h  Menu.cpp  Menu.h  Player.cpp  Source.cpp*  Person.cp
Game.v1
9
10 //setters
11 void SetScore(int score);
12 void SetScoreboardPosition(string scoreboardPosition);
13
14
15 //Getters
16 int GetScore();
17 string GetScoreboardPosition();
18
19 void PrintPerson();
20
21
22 private:
23     int _score;
24     string _scoreboardPosition;
25     };
```

Figure 6 Screenshot to show the use of private attributes and accessible methods

Sectional Word Count (1170)

Conclusion

In conclusion we can see that through the implementation of key object-oriented programming concepts we can create multiple classes which can be linked together, through these links we can illustrate a 'real world' situation such as that of a person being a player that plays a game of tic-tac-toe. Furthermore, this approach to programming enables developers to break a larger problem down into smaller more manageable ones. By doing so problems naturally become easier to solve and test therefore making it easier to isolate errors. Finally, through the use of inheritance and polymorphism there is less need to duplicate code, which ensures the program is easier for others to read.

Sectional word count (114)

Total word Count (1444/3000)

References

- [1] W3schools, n.d. *W3schools*. [Online]
Available at: https://www.w3schools.com/cpp/cpp_polymorphism.asp
[Accessed 28th October 2021].

- [2] J, Hartman, (7th October 2021) What is Abstraction in OOPs? [Online]
Available at: <https://www.guru99.com/java-data-abstraction.html>
[Accessed 21 November 2021].

Testing

A sample of the tests performed to evaluate the programme during and after the development stage

Test number	Purpose of the test	Input	Expected screen output	Pass or Fail
1.	Test on Person constructors			
1.1	Tests that a new Person object can be created using the default constructor	<pre>Person* person1 = new Person(); cout << person1 << endl;</pre>	To print out the current memory address of the object	Pass
1.2	Tests the parameterized constructor	<pre>Person* person2 = new Person("James", 12, "Scotland"); cout << "The players name is: " << person2->GetName() << endl; cout << "The players age is: " << person2->GetAge() << endl; cout << "The players is located in: " << person2->GetLocation() << endl;</pre>	The players name is: James The players age is: 12 The players is located in: Scotland	Pass

Test number	Purpose of the test	Input	Expected screen output	Pass or Fail
2	Tests the setters and getter methods			
2.1	SetName() and GetName()	<pre>person1->SetName("Freddie"); cout << "The players name is: " << person1->GetName() << endl;</pre>	The players name is: Freddie	Pass
2.2	SetAge() and GetAge()	<pre>person1->SetLocation("Wales"); cout << "The players age is: " << person1->GetAge() << endl;</pre>	The players age is: 7	Pass
2.3	SetLocation() and GetLocation()	<pre>person1->SetAge(7); cout << "The players is located in: " << person1->GetLocation() << endl;</pre>	The players is located in: Wales	Pass
3	printPerson() virtual method Test 1a: Purpose: to create a Person object and show that the compiler will only print the name, age and location assigned to that object Test 1b: Purpose: to create a player object and show that the compiler will print out the name, age and location as before but also print out the score and scoreboard position assigned to that object	<pre>Person* player4 = new Person(); >SetName("Freddie"); >SetLocation("Wales"); player4->SetAge(7); player4->PrintPerson(); cout << "\n\n\n"; Player* player5 = new Player(); >SetName("James"); >SetLocation("Wales"); player5->SetAge(17); player5->SetScore(6);</pre>	The persons details are as follows Name: Freddie Age: 7 Geographical location: Wales The players details are as follows Name: James Age: 17 Geographical location: Wales Their current score is 6, which puts them at scoreboard position : First.	Pass

Test number	Purpose of the test	<code>player5->SetScoreboardPosition("First");</code> <code>player5->PrintPerson();</code>	Expected screen output	Pass or Fail
4	Test the main menu options A series of tests to ensure that each option of the main menu performs as required in both valid entries and invalid entries from the user.	Input		
4.1	displayMenu()	Code execution: Menu* menu1 = new Menu(); menu1->DisplayMenu();	Main menu for the game displays.	Pass
4.2	0 Quit	Code execution: Menu* menu1 = new Menu(); menu1->DisplayMenu(); user input: 0	Main menu for the game displays Game ends and console closes	Pass
4.3	1 Play Game	Code execution: Menu* menu1 = new Menu(); menu1->DisplayMenu(); user input: 1	Message output: Cool, lets get ready to play.....	Pass
4.4	2 Scoreboard	Code execution: Menu* menu1 = new Menu(); menu1->DisplayMenu(); User input: 2	The console will output the scoreboard	Pass

4.5	3 Rules	Code execution: <pre>Menu* menu1 = new Menu(); menu1->DisplayMenu();</pre> User input: 3	Message output: The rules are..... Additional action: the sub menu will be displayed to enable the user to go back to the main menu	Pass
4.6	Invalid menu selection	Code execution: <pre>Menu* menu1 = new Menu(); menu1->DisplayMenu();</pre> user input: 7	Message output: Sorry, but 7 isn't a valid choice, please choose a valid menu selection. Action: the menu will display again and the user will be prompted to enter a valid choice.	Pass
5	Test the sub menu options			
5.1	0 Quit	Code execution: <pre>Menu* menu1 = new Menu(); menu1->DisplayMenu();</pre> user input : 0	Game ends and console closes	Pass
5.2	1 display main menu	Code execution: <pre>Menu* menu1 = new Menu(); menu1->DisplayMenu();</pre> User input: 1	Main menu for the game displays.	Pass

Tests on creating a player

Test number	Purpose of the test	Input	Expected screen output	Pass or Fail
6.1	To test that a player can be created using valid data from the user input	James 6 Swansea	Enter the name of player 2 please.....	Pass
6.2	To test that an invalid entry for the players age will not allow a player object to be created.	Thomas Y Wales	The players age must be entered as a numerical value ie 2 or 56. Please enter the players age.....	Pass

Tests carried out during game play

Test Number	Testing possible win combinations and incrementation of the players score	Input for player X to win	Input for player O to win	output	Pass/Fail
7	123 – top row	T1 –1 T2-2 T3- 3	T1 –4 T2- 5	Player 1, James is the winner!!! Your new score is 1 Also shows scoreboard incrementation	Pass – see figure 6
8	456 – middle row	T1 –1 T2-2 T3- 9	T1 –4 T2-5 T3- 6	Player 2, Thomas is the winner!!! Your new score is 1	Pass – see figure 7

					Also shows scoreboard incrementation	
9	789 – bottom row	T1 –7 T2-8 T3- 9	T1 –4 T2- 5	Player 1, James is the winner!!! Your new score is 2	Pass – see figure 8	
10	147 – left column	T1 –1 T2-4 T3- 7	T1 –6 T2- 5	Also shows scoreboard incrementation Player 1, James is the winner!!! Your new score is 3	Pass – see figure 9	
11	258 – middle column	T1 –1 T2-4 T3- 9	T1 –2 T2-5 T3- 8	Also shows scoreboard incrementation Player 2, Thomas is the winner!!! Your new score is 2	Pass	
12	369 – right column	T1 –1 T2-4 T3- 5	T1 –3 T2-6 T3- 9	Player 1, James is the winner!!! Your new score is 4	Pass	
13	159 – diagonal		T1 –1 T2-5 T3- 9	Player 2, Thomas is the winner!!! Your new score is 3	Pass	

14	753 - diagonal	T1 -7 T2-5 T3- 3	T1 -1 T2-4 T3- 5	Player 1, James is the winner!!! Your new score is 5	Pass
----	----------------	------------------------	------------------------	--	------

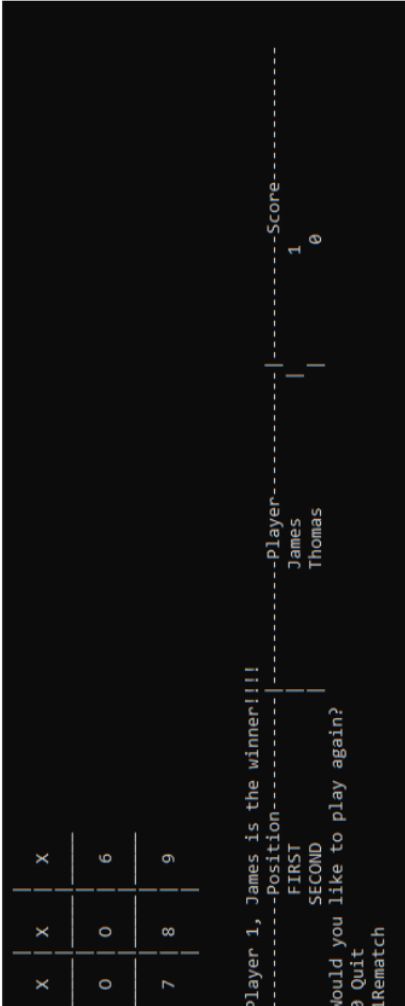


Figure 7 Screen shot of test number 7

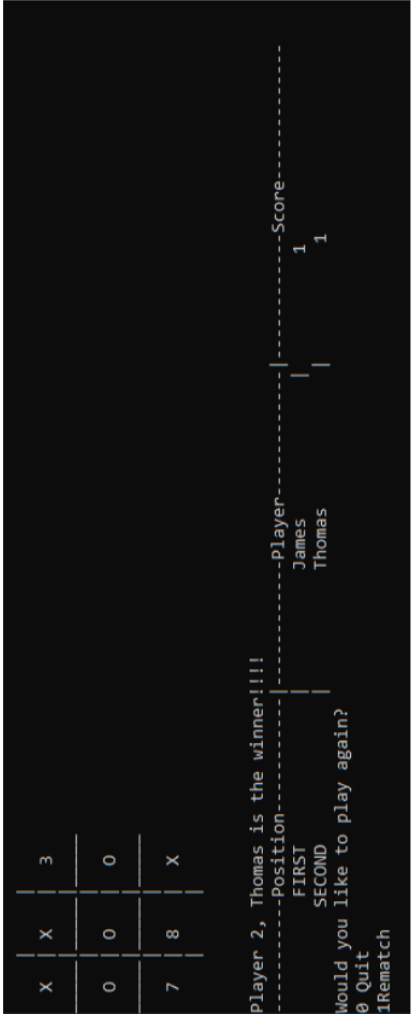


Figure 8 screen snip of test number 8



Figure 9 screen snip to show test number 9



Figure 10 Screen snip to show test 10

Appendix

Video Link

Link to youtube video of the game run through

https://youtu.be/nbQY_I6obZ0

Link to youtube video on key concepts

<https://youtu.be/PHHnYnZyfwI>

Code

Person.cpp

```
#include "Person.h"

Person::Person()
{
    _age = 0;
}

Person::Person(string name, int age, string location)
{
    _name = name;
    _age = age;
    _location = location;
}

void Person::SetName(string name)
{
    _name = name;
}

void Person::SetAge(int age)
{
    _age = age;
}

void Person::SetLocation(string location)
{
    _location = location;
}

string Person::GetName()
{
    return _name;
}
```

```
int Person::GetAge()
{
    return _age;
}

string Person::GetLocation()
{
    return _location;
}

void Person::PrintPerson()
{
    cout << "The persons details are as follows" << endl << "Name: " << GetName()
<< endl << "Age: " << GetAge() << endl << "Geographical Location: " << GetLocation();
}
```

Person.h

```
#pragma once
#include <iostream>

using namespace std;

class Person
{
public:
    Person();
    Person(string name,int age, string location);

    //setters
    void SetName(string neme);
    void SetAge(int age);
    void SetLocation(string location);

    //Getters
    string GetName();
    int GetAge();
    string GetLocation();

    virtual void PrintPerson();

private:
    string _name;
    int _age;
    string _location;
};
```

Person.cpp

```
#include "Player.h"

Player::Player()
{
    _score = 0;
    _scoreboardPosition = "none";
}

Player::Player(int score, string scoreboardPosition)
{
    _score = score;
    _scoreboardPosition = scoreboardPosition;
}

void Player::SetScore(int score)
{
    _score = score;
}

void Player::SetScoreboardPosition(string scoreboardPosition)
{
    _scoreboardPosition = scoreboardPosition;
}

int Player::GetScore()
{
    return _score;
}

string Player::GetScoreboardPosition()
{
    return _scoreboardPosition;
}

void Player::PrintPerson()
{
    cout << "The players details are as follows" << endl <<
        "Name: " << GetName() << endl << "Age: " << GetAge() << endl <<
```

```
        "Geographical Location: " << GetLocation()<< endl <<
        "Their current score is: " << GetScore() << ", which puts them at
scoreboard position: " << GetScoreboardPosition();
    }
```

Player.h

```
#pragma once
#include "Person.h"

class Player: public Person
{
public:
    Player();
    Player(int score, string scoreboardPosition);

    //setters
    void SetScore(int score);
    void SetScoreboardPosition(string scoreboardPosition);

    //Getters
    int GetScore();
    string GetScoreboardPosition();

    void PrintPerson();

private:
    int _score;
    string _scoreboardPosition;
};
```

Menu.cpp

```
#include "Menu.h"
```

```
#include "Person.h"
```

```
#include "Player.h"
```

```
Menu::Menu()
```

```
{
```

```
    //default constructor
```

```
}
```

```
void Menu::DisplayMenu() // displays the main menu
```

```
{
```

```
    int choice;
```

```
    cout << "Welcome to the Tic-Tak-Toe 3 in a row Game!" << endl;
```

```
    cout << endl;
```

```
    cout << "Menu!!" << endl;
```

```
    cout << "0 Quit" << endl;
```

```
    cout << "1 Play Game" << endl;
```

```
    cout << "2 Score Board" << endl;
```

```
    cout << "3 Rules" << endl;
```

```
    cout << endl;
```

```
    cout << "Please enter a valid menu selection....." << endl;
    cin >> choice;

    switch (choice)
    {
    case 0:
        Quit();
        break;
    case 1:
        cout << "Cool, let's get ready to play!" << endl;
        cout << endl;
        drawBoard();
        break;
    case 2:
        scoreboard();
        break;
    case 3:
        cout << "The rules are....." << endl;
        DisplayRules();
        break;
    default:
        cout << "\nSorry, but " << choice << " isn't a valid choice, please choose a
valid menu selection.\n";
        cout << "\n";
        DisplayMenu();
    }while (choice < 4); // ensures that the user inputs a valid menu choice
}

void Menu::subMenu()// produces a sub-menu for the player to navigate back to the main
menu
{
    int input;
    cout << "Please select one of the following options....." << endl;
    cout << "0 Quit" << endl;
    cout << "1 Return to main menu" << endl;
```

```
    cin >> input;
    switch (input)
    {
    case 0:
        Quit();
        break;
    case 1:
        DisplayMenu();
        break;
    default:
        cout << "\nSorry, but " << input << " isn't a valid choice.\n";
        subMenu();
    }while (input < 2); // ensures that the user inputs a valid menu choice
}

void Menu::DisplayRules() // displays the rules of the game
{
    cout << "Rule 1: \n" << "Players must place three of their counter in a row to win" << endl;
    cout << endl;
    cout << "Rule 2: \n" << "Players can also win by placing 3 of their counter diagonally" << endl;
    cout << endl;
    cout << "Rule 3: \n" << "Only one piece is played at a time and players must take turns." << endl;
    cout << endl;
    cout << "Rule 4: \n" << "Players can be on the offensive or defensive.." << endl;
    cout << endl;
    cout << "Rule 5: \n" << "The game ends when there is a 3-in-a-row or a stalemate.." << endl;
    cout << endl;
    cout << endl;

    subMenu();
}
```

```
void Menu::Quit() //closes the game down
{
    cout << "See you again soon....." << endl;
    delete player1;
    delete player2;
    exit(0);
}

void Menu:: drawBoard() // creates the board
{
    createPlayer();

    cout << "So " << player1->GetName() << " and " << player2->GetName() << " are you
ready to play" << endl;
    cout << "\n\n\n\n";

    cout << "PLAYER - 1  [X]  PLAYER - 2  [O]\n";
    cout << "\n";
    cout << "\n";

    cout << "      |      |      \n";
    cout << " " << board[0][0] << " | " << board[0][1] << " | " << board[0][2] <<
" \n";
    cout << "____|____|____\n";
    cout << "      |      |      \n";
    cout << " " << board[1][0] << " | " << board[1][1] << " | " << board[1][2] <<
" \n";
    cout << "____|____|____\n";
    cout << "      |      |      \n";
    cout << " " << board[2][0] << " | " << board[2][1] << " | " << board[2][2] <<
" \n";
    cout << "      |      |      \n";
    cout << "\n";
```

```

        cout << "\n";

        turn();
    }

void Menu::updateBoard() //updates the board after each player takes their turn
{

    cout << "\n";
    cout << "\n";
    cout << "      |      |      \n";
    cout << " " << board[0][0] << " | " << board[0][1] << " | " << board[0][2] <<
    "\n";
    cout << "____|____|____\n";
    cout << "      |      |      \n";
    cout << " " << board[1][0] << " | " << board[1][1] << " | " << board[1][2] <<
    "\n";
    cout << "____|____|____\n";
    cout << "      |      |      \n";
    cout << " " << board[2][0] << " | " << board[2][1] << " | " << board[2][2] <<
    "\n";
    cout << "      |      |      \n";
    cout << "\n";
    cout << "\n";
}

void Menu::resetBoard() // reset the board if the players wish to play again
{
    cout << "\nSo " << player1->GetName() << " and " << player2->GetName() << " let's
    have a rematch\n";

    cout << "\n\n\n\n";
    cout << "PLAYER - 1  [X]  PLAYER - 2  [O]\n";
    cout << "\n";
    cout << "\n";
    cout << "      |      |      \n";
    cout << " " << (board[0][0] = '1') << " | " << (board[0][1] = '2') << " | "
    << (board[0][2] = '3') << "\n";
}

```

```
    cout << " ____|____|____\n";
    cout << "      |      |      \n";
    cout << "  " << (board[1][0]= '4') << "  |  " << (board[1][1]='5') << "  |  " <<
(board[1][2]='6') << " \n";
    cout << " ____|____|____\n";
    cout << "      |      |      \n";
    cout << "  " << (board[2][0]= '7') << "  |  " << (board[2][1]='8') << "  |  " <<
(board[2][2]='9') << " \n";
    cout << "      |      |      \n";
    cout << "\n";
    cout << "\n";

    token = 'X';
    draw;
    winner = false;
    isGameOver = true;
    cout << player1->GetScore() << endl;
    cout << player2->GetScore() << endl;

    turn();
}
```

```
void Menu::playAgain() //allows the players to choose to play again after a game has
ended
```

```
{
    int input;
    int playagain;

    if (winner == true)
    {
        cout << "Would you like to play again?" << endl;
        cout << "0 Quit" << endl;
        cout << "1Rematch" << endl;
        cin >> input;
```

```
        switch (input)
        {
        case 0:
            Quit();
            break;
        case 1:
            cout << "\nCool, let's get ready to play!\n";
            resetBoard();
            cout << endl;

            break;
        default:
            cout << "\nSorry, but " << input << " isn't a valid choice.\n";
        }while (input < 2); // ensures that the user inputs a valid menu choice
    }
}

void Menu::scoreboard() //produces a scoreboard
{
    string position1;
    string position2;
    int currentScore1;
    int currentScore2;

    player1->SetScoreboardPosition("First");
    position1 = player1->GetName();
    currentScore1 = player1->GetScore();
    player2->SetScoreboardPosition("Second");
    position2 = player2->GetName();
    currentScore2 = player2->GetScore();

    cout << "|" << "-----Player-----" << " | " << "-----Score----"
    ----- << "\n";
```

```
        cout << "|" << "          "<< position1 << "          "<< "|" << "
"<< currentScore1 << "\n";

        cout << "|" << "          "<< position2 << "          "<< "|" << "
"<< currentScore2 << "\n";

    }

void Menu::createPlayer() //creates two players
{
    string n1;
    int a1;
    string l1;

    cout << "Enter the name of player 1 please....." << endl;
    cin >> n1;
    cout << "Enter the age of player 1 please....." << endl;
    cin >> a1;
    while (cin.fail()) // ensures that the user enters an integer value for the age.
    {
        cin.clear();
        cin.ignore();

        cout << "The players age must be entered as a numerical value ie 2 or 56.
Please enter the players age....." << endl;
        cin >> a1;
    }

    cout << "Enter the geographical location of player 1 please....." << endl;
    cin >> l1;

    player1->SetName(n1);
    player1->SetAge(a1);
    player1->SetLocation(l1);

    cout << endl;
```

```
    string n2;
    int a2;
    string l2;

    cout << "Enter the name of player 2 please....." << endl;
    cin >> n2;
    cout << "Enter the age of player 2 please....." << endl;
    cin >> a2;
    while (cin.fail()) // ensures that the user enters an integer value for the age.
    {
        cin.clear();
        cin.ignore();
        cout << "The players age must be entered as a numerical value ie 2 or 56.
Please enter the players age....." << endl;
        cin >> a2;
    }
    cout << "Enter the geographical location of player 2 please....." << endl;
    cin >> l2;

    player2->SetName(n2);
    player2->SetAge(a2);
    player2->SetLocation(l2);

    cout << "create player" << endl;
}

void Menu::turn() // prompts each player to place a counter in turn
{
    int choice;
    counter = 0;
    cout << counter;

    //player1 plays first
```

```
while (winner == false)
{
    if (token == 'X')
    {
        cout << player1->GetName() << " please enter the square number you wish to
place a counter" << endl;
        cin >> choice;

    }
    else if (token == 'O')
    {
        cout << player2->GetName() << " please enter the square number you wish to
place a counter" << endl;
        cin >> choice;

    }
    if (choice >= 1 && choice <= 9) {
        switch (choice)
        {
            case 1:
                row = 0; column = 0;
                break;
            case 2:
                row = 0; column = 1;
                break;
            case 3:
                row = 0; column = 2;
                break;
            case 4:
                row = 1; column = 0;
                break;
            case 5:
                row = 1; column = 1;
                break;
            case 6:
```

```
        row = 1; column = 2;
        break;
    case 7:
        row = 2; column = 0;
        break;
    case 8:
        row = 2; column = 1;
        break;
    case 9:
        row = 2; column = 2;
    }
}
else
{
    cout << "please enter a valid square number between 1 and 9 to place a
counter" << endl;
    cout << token;
}

if (token == 'X' && board[row][column] != 'X' && board[row][column] != 'O')
{
    board[row][column] = 'X';
    updateBoard();
    win();
    counter = (counter + 1);
    cout << counter << " counter" << endl;
    token = 'O';
}
else if (token == 'O' && board[row][column] != 'X' && board[row][column] !=
'0')
{
    board[row][column] = 'O';
    updateBoard();
    win();
    counter = (counter + 1);
    cout << counter << " counter" << endl;
```

```
        token = 'X';
    }
    gameOver();
}
}
```



```
void Menu::gameOver() //stops the game if there are no winners
{
    while (counter == 9)
    {
        isGameOver = false;
        int choice;

        cout << "\n*****game
over*****\n";

        cout << "\n The game is drawn, if you would like to play again? Select 1 to
play again or 0 to exit the game\n";

        cout << "\n0 quit\n";
        cout << "\n1 play game\n";
        cin >> choice;

        switch (choice)
        {
        case 0:
            Quit();
            break;

        case 1:
            cout << "cool, let's get ready to play!" << endl;
            cout << endl;
            drawBoard();
            break;

        default:
            cout << "\nsorry, but " << choice << " isn't a valid choice, please choose
a valid menu selection.\n";
            cout << "\n";
            scoreboard();
        }
    }
}
```

```
        playAgain();
        while (choice < 2); // ensures that the user inputs a valid menu choice
    }

}

}

void Menu::win() // checks for three counters in a row from one player in a variety of
combinations
{
    //1|2|3   = //00|01|02
    //4|5|6   = //10|11|12
    //7|8|9   = //20|21|22

    //checks for a horizontal win (1,2,3)
    if (board[0][0] == token && board[0][1] == token && board[0][2] == token)
    {
        if (token == 'X')
        {
            player1->SetScore(player1->GetScore() + 1);
            cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
        }
        else
        {
            player2->SetScore(player2->GetScore() + 1);
            cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
        }
        winner = true;
        scoreboard();
        playAgain();
    }

    //checks for a horizontal win (4,5,6)
    if (board[1][0] == token && board[1][1] == token && board[1][2] == token)
```

```
{

    if (token == 'X')
    {
        player1->SetScore(player1->GetScore() + 1);
        cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
    }
    else
    {
        player2->SetScore(player2->GetScore() + 1);
        cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
    }
    winner = true;
    scoreboard();
    playAgain();
}

//checks for a horizontal win (7,8,9)
if (board[2][0] == token && board[2][1] == token && board[2][2] == token)
{

    if (token == 'X')
    {
        player1->SetScore(player1->GetScore() + 1);
        cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
    }
    else
    {
        player2->SetScore(player2->GetScore() + 1);
        cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
    }
    winner = true;
    scoreboard();
}
```

```
        playAgain();
    }

    //checks for a Vertical win (1,4,7)

    if (board[0][0] == token && board[1][0] == token && board[2][0] == token)
    {

        if (token == 'X')
        {
            player1->SetScore(player1->GetScore() + 1);
            cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
        }
        else
        {
            player2->SetScore(player2->GetScore() + 1);
            cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
        }
        winner = true;
        scoreboard();
        playAgain();
    }

    //checks for a Vertical win (2,5,8)

    if (board[0][1] == token && board[1][1] == token && board[2][1] == token)
    {

        if (token == 'X')
        {
            player1->SetScore(player1->GetScore() + 1);
            cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
        }
    }
```

```
        else
        {
            player2->SetScore(player2->GetScore() + 1);
            cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
        }
        winner = true;
        scoreboard();
        playAgain();
    }
    //checks for a Vertical win (3,6,9)
    if (board[0][2] == token && board[1][2] == token && board[2][2] == token)
    {

        if (token == 'X')
        {
            player1->SetScore(player1->GetScore() + 1);
            cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;

        }
        else
        {
            player2->SetScore(player2->GetScore() + 1);
            cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
        }
        winner = true;
        scoreboard();
        playAgain();
    }

    //Checks for a diagonoal win (1,5,9)
    if (board[0][0] == token && board[1][1] == token && board[2][2] == token)
    {
```

```
        if (token == 'X')
        {
            player1->SetScore(player1->GetScore() + 1);
            cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
        }
        else
        {
            player2->SetScore(player2->GetScore() + 1);
            cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
        }
        winner = true;
        scoreboard();
        playAgain();
    }
    //Checks for a diagonal win (7,5,3)
    if (board[2][0] == token && board[1][1] == token && board[0][2] == token)
    {

        if (token == 'X')
        {
            player1->SetScore(player1->GetScore() + 1);
            cout << "Player 1, " << player1->GetName() << " is the winner!!!! Your new
score is " << player1->GetScore() << endl;
        }
        else
        {
            player2->SetScore(player2->GetScore() + 1);
            cout << "Player 2, " << player2->GetName() << " is the winner!!!! Your new
score is " << player2->GetScore() << endl;
        }
        winner = true;
        scoreboard();
        playAgain();
    }
```

```
    }  
  
}
```

Menu.h

```
#pragma once  
#include <iostream>  
#include "Player.h"  
  
using namespace std;  
  
class Menu:public Player  
{  
    char place[6][7];  
    Player* player1 = new Player();  
    Player* player2 = new Player();  
    char board[3][3] = { {'1','2','3'},{'4','5','6'},{'7','8','9'} };  
    int row;  
    int column;  
  
public:  
    Menu();//default constructor  
    void DisplayMenu();  
    void subMenu();  
    void scoreboard();  
    void DisplayRules();  
    void Quit();  
  
    void createPlayer();  
    void drawBoard();  
    void updateBoard();  
    void resetBoard();  
    void playAgain();  
  
    void turn();  
    void move();  
    void gameOver();  
    void win();  
  
private:  
    char token = 'X';  
    bool draw;  
    int counter = 0;  
    bool winner = false;  
    bool isGameOver = true;
```

```
};
```

Source.cpp

```
#include <string>
#include <iostream>           // For cin, cout and getline()
#include <sstream>
#include "Person.h"
#include "Player.h"
#include "Menu.h"
#include <Windows.h>

using namespace std;        // Add this before the main

int main(int argc, char* argv[])
{
    Menu* menu1 = new Menu();
    menu1->DisplayMenu();

    //-----
    //Test code to show that virtual method can be overridden from class Person by
    class player.
    //Person* player4 = new Person();
    //player4->SetName("Freddie");
    //player4->SetLocation("Wales");

    //player4->PrintPerson();

    //cout << "\n\n\n";

    //Player* player5 = new Player();
    //player5->SetName("James");
    //player5->SetLocation("Wales");
    //player5->SetAge(17);
    //player5->SetScore(6);
    //player5->SetScoreboardPosition("First");

    //player5->PrintPerson();

    //-----
```

```
//Test to show that an object of class Person can be created using the default
constructor
//Person* person1 = new Person();
//cout << person1 << endl;

//-----

//Tests the parameterized constructor
//Person* person2 = new Person("James", 12, "Scotland");
//cout << "The players name is: " << person2->GetName() << endl;
//cout << "The players age is: " << person2->GetAge() << endl;
//cout << "The players is located in: " << person2->GetLocation() << endl;

//-----

//Tests the Setter methods in class Person
//person1->SetName("Freddie");
//person1->SetLocation("Wales");
//person1->SetAge(7);

//-----

//Tests the getter methods in class person
//cout << "The players name is: " << person1->GetName() << endl;
//cout << "The players age is: " << person1->GetAge() << endl;
//cout << "The players is located in: " << person1->GetLocation() << endl;

//-----

//Test to show that an object of class Player can be created using the default
constructor
//Player* player0 = new Player();

//-----

//Tests the setter methods for class Player
//player0->SetScore(12);
//player0->SetScoreboardPosition(15);
//cout << "The players score is: " << player0->GetScore() << endl;
//cout << "The player is at scoreboard position: " << player0-
>GetScoreboardPosition() << endl;

//-----

//Test to show that an object of class Player can be created using the
parameterized constructor
//Player* player1 = new Player(6,20);
//cout << player1 << endl;
//Tests the getter methods
//cout << "The players score is: " << player1->GetScore() << endl;
//cout << "The player is at scoreboard position: " << player1-
>GetScoreboardPosition() << endl;
//player1->SetName("Thomas");
//player1->SetAge(12);
//player1->SetLocation("Hamburg");
//cout << endl;

//player1->PrintPerson();
//cout << endl;
```

```
    //-----  
    // Tests that an object of type Player can be created using the Parent class  
Person.  
    //Person* player2 = new Player(6,15);  
    //cout << player2 << endl;  
  
    //player2->SetName("james");  
    //player2->SetAge(16);  
    //player2->SetLocation("Swansea");  
    //player2->PrintPerson();  
  
    //Player* player3 = new Player();  
  
    //delete player0;  
    //delete player1; // objects player1 and player2 deleted in the menu.cpp  
    //delete player5;  
    //delete player4;  
    //delete menu1;  
  
    // Pause  
    cin.get();          // Add before ♦return♦__  
    return 0;  
}
```

FINAL GRADE

72/100

GENERAL COMMENTS

Instructor

Classes (25): 20
Polymorphism (10): 0

Game (20): 15

Report (35): 28

Testing (5): 5

Demo (5): 4

Feedback:
=====

Report could have clarified a couple of points.

Noughts and crosses game. The classes are little limited. I am not convinced that Menu class should inherit from Person. Also, this class really needed to be split up into a Game and a Menu class

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6



Comment 1

Not quite. An important point is that the attributes are like global variables but ring fenced by the class...they are hidden.

PAGE 7

PAGE 8



Comment 2

How does that look in the code?

PAGE 9



Comment 3

Abstraction is bit more interesting than this, what you have described is data hiding. Also why does abstraction reduce complexity?

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47