Experiment 5
Inheritance

Objectives

To create classes by inheriting from existing classes.
The notions of base classes and derived classes and the relationships between them.

Prelab Activities

CorrecttheCode

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, write the corrected code. If the code does not contain an error, write "no error." For code segments, assume the code appears in main and that using directives are provided. [Note: It is possible that a program segment may contain multiple errors.]

1. Class X inherits from class Y.

```
1  #include <iostream>
2  using namespace std;
3
4  // class Y definition
5  class Y
6  {
7  public:
8     Y(); // default constructor
9     ~Y(); // destructor
10 private:
11    int data;
12 }; // end class Y
13
14 // class X definition
15 class X ; public Y
16 {
17 public:
18    // function print
19    void print() const
20    {
21       cout << data;
22    } // end function print
23 }; // end class X
```

2. The following code should construct a Derived object.

```
1  #include <iostream>
2  using namespace std;
3
4  // class Base definition
```

```
5  class Base
6  {
7  private:
8     // constructor
9     Base( int b )
10    {
11       cout << b;
12    } // end class Base constructor
13 }; // end class Base
14
15 // class Derived definition
16 class Derived : public Base
17 {
18    // constructor calls base-class constructor
19    Derived( int a )
20       : Base( a )
21    {
22       // empty
23    } // end class Derived constructor
24 }; // end class Derived
25
26 int main()
27 {
28    Derived d( 5 );
29 } // end main
```

3. The following code creates an object of type B. Class B inherits from class A.

```
1  #include <iostream>
2  using namespace std;
3
4  // class A definition
5  class A
6  {
7  public:
8     // constructor
9     A( int a )
10    {
11       value = a;
12    } // end class A constructor
13
14    // return value
15    int getValue() const
16    {
17       return value;
18    } // end function getValue
19 private:
20    int value;
21 }; // end class A
22
23 // class B definition
24 class B
25 {
26 public:
27    // constructor
28    B( int b )
29       : A( b )
30    {
31       // empty
32    } // end class B constructor
```

```
33 }; // end class B
34
35 int main()
36 {
37    B object( 50 );
38    cout << object.getValue();
39 } // end main
```

4. The following code creates an object of type Y. Class Y inherits from class X.

```
1  #include <iostream>
2  using namespace std;
3
4  // class X definition
5  class X
6  {
7  public:
8     // constructor
9     X()
10    {
11       cout << "X constructed!";
12    } // end class X constructor
13 }; // end class X
14
15 // class Y definition
16 class Y
17 {
18 public:
19    // redefine inherited constructor
20    X()
21    {
22       cout << "Y created, not X!";
23    } // end class Y constructor
24 }; // end class Y
25
26 int main()
27 {
28    Y yObject();
29 } // end main
```

Lab Exercises

Lab Exercise 1 — Football Players

The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. UML Diagrams
4. Sample Output
5. Test Code
6. Problem-Solving Tips

The test code represents a complete working C++ test program Read the problem description and examine the sample output; then study the test code. Using the problem-solving tips as a guide, write your C++ code. Compile and execute the program. Compare your output with the sample output provided.

Lab Objectives

In this lab, you will practice:

- Using inheritance to create a football player hierarchy that includes a Player class, a FootballPlayer class, a GoalKeeper class, DefensivePlayer class, a MiddleFielderPlayer class and a ForwardPlayer class.
- Using private data members to limit access to data members.

Description of the Problem

The class hierarchy represent a football player team hierarchy. Every player belongs to a specific position and every one of them has specific duties in the play. Your job is to implement this hierarchy and specific duties, using object oriented programming techniques (inheritance).

UML Diagrams

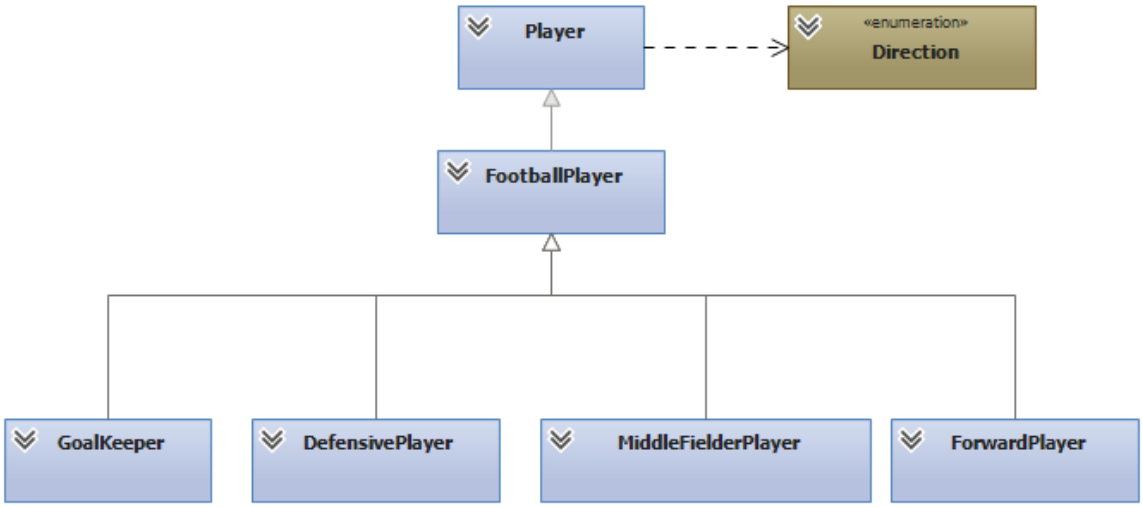

Figure 1:General UML Diagram

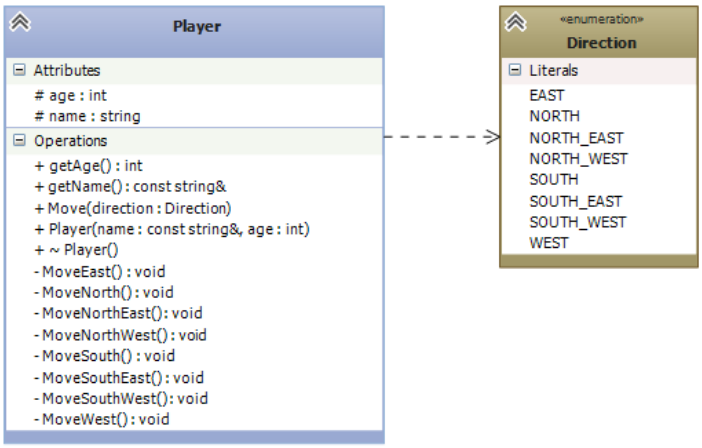

Figure 2:Player class

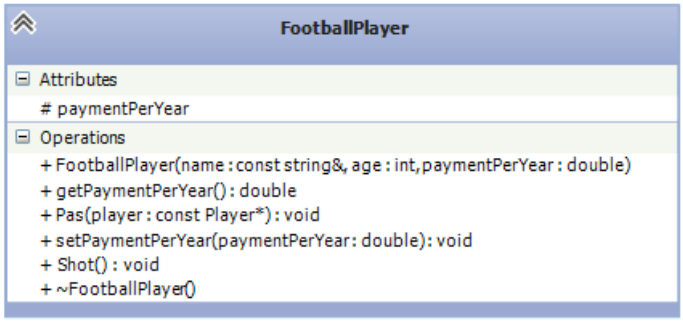

Figure3:FootballPlayer class

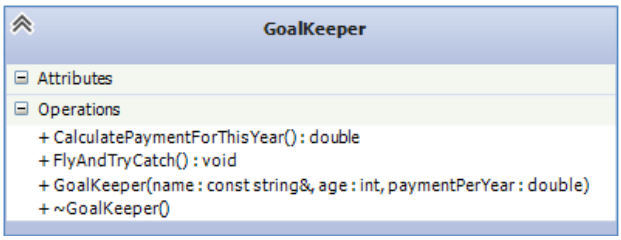

Figure 4:GoalKeeper class

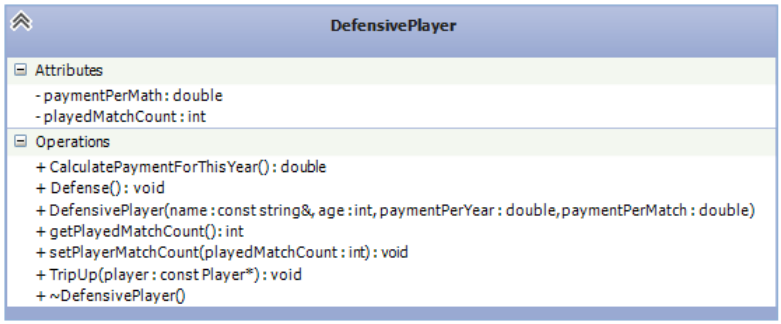

Figure 5:DefensivePlayer class

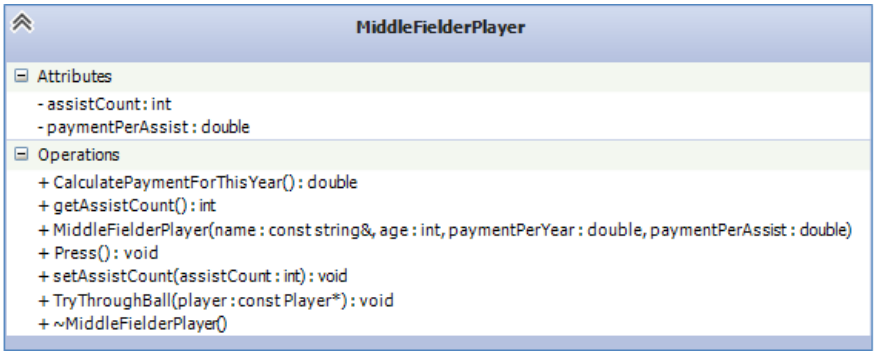

Figure 6:MiddleFielderPlayer

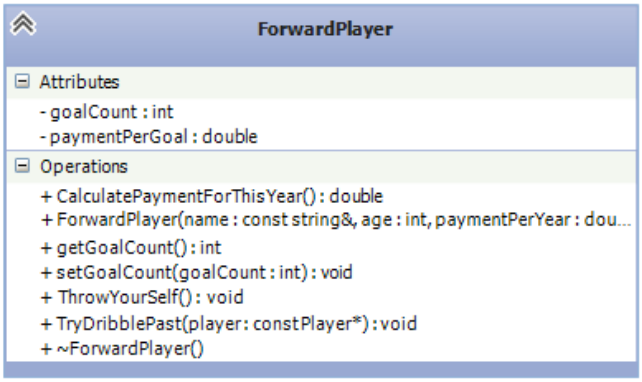

Figure 7:ForwardPlayer class

Sample Output

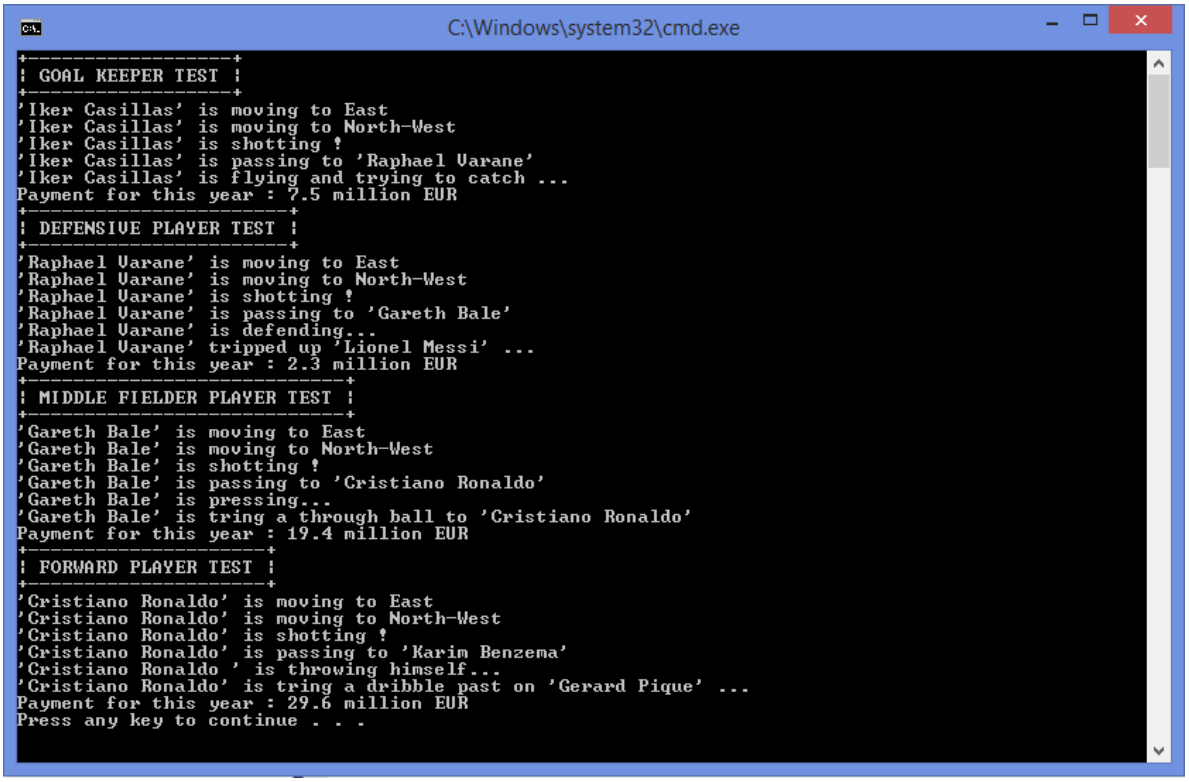

Figure 8:Sample Output

Test Code (You are not allowed to change the test code)

```
/****************************************
*****************************************
* IDE : Visual Studio 2015              *
* Author : Cihan UYANIK                 *
* Experiment 5: Inheritance             *
*****************************************/


#include <iostream>
#include "GoalKeeper.h"
#include "DefensivePlayer.h"
#include "MidFielderPlayer.h"
#include "ForwardPlayer.h"
using namespace std;

void TEST_GoalKeeper()
{
	cout <<"+------------------+" << endl
		<< "| GOAL KEEPER TEST |" << endl
		<< "+------------------+" << endl;

	GoalKeeper* keeper = new GoalKeeper("Iker Casillas", 34, 7.5);
	keeper->Move(Player::Direction::EAST);
	keeper->Move(Player::Direction::NORTH_EAST);
	keeper->Shot();
```

```
	DefensivePlayer* player = new DefensivePlayer("Raphael Varane", 22, 1.2, 0.1);
	keeper->Pas(player);

	keeper->FlyAndTryCatch();

	cout << "Payment for this year : " << keeper->CalculatePaymentForThisYear()
<<" million EUR"<<endl;
}

void TEST_DefensivePlayer()
{
	cout <<"+-----------------------+" << endl
		<< "| DEFENSIVE PLAYER TEST |" << endl
		<< "+-----------------------+" << endl;
	DefensivePlayer* defense = new DefensivePlayer("Raphael Varane", 22, 1.2,
0.1);
	defense->Move(Player::Direction::EAST);
	defense->Move(Player::Direction::NORTH_EAST);

	defense->Shot();

	MidFielderPlayer* player = new MidFielderPlayer("Gareth Bale", 25, 15, 0.2);
	defense->Pas(player);

	defense->Defense();
	ForwardPlayer* opponent = new ForwardPlayer("Lionel Messi", 28, 20, 0.3);
	defense->TripUp(opponent);

	defense->setPlayedMatchCount(11);
	cout << "Payment for this year : " << defense->CalculatePaymentForThisYear()
<< " million EUR" << endl;
}

void TEST_MiddleFielderPlayer()
{
	cout <<"+----------------------------+" << endl
		<< "| MIDDLE FIELDER PLAYER TEST |" << endl
		<< "+----------------------------+" << endl;
	MidFielderPlayer* middleFielder = new MidFielderPlayer("Gareth Bale", 25, 15,
0.2);
	middleFielder->Move(Player::Direction::EAST);
	middleFielder->Move(Player::Direction::NORTH_EAST);

	middleFielder->Shot();

	ForwardPlayer* player = new ForwardPlayer("Cristiano Ronaldo", 30, 17, 0.3);
	middleFielder->Pas(player);

	middleFielder->Press();
	middleFielder->TryThroughBall(player);

	middleFielder->setAssistCount(22);
	cout << "Payment for this year : " << middleFielder-
>CalculatePaymentForThisYear() << " million EUR" << endl;
}
```

```
void TEST_ForwardPlayer()
{
	cout <<"+---------------------+" << endl
		<< "| FORWARD PLAYER TEST |" << endl
		<< "+---------------------+" << endl;

	ForwardPlayer* forward = new ForwardPlayer("Cristiano Ronaldo", 30, 17, 0.3);
	forward->Move(Player::Direction::EAST);
	forward->Move(Player::Direction::NORTH_EAST);

	forward->Shot();

	ForwardPlayer* player = new ForwardPlayer("Karim Benzema", 27, 8, 0.3);
	forward->Pas(player);

	forward->ThrowYourSelf();

	DefensivePlayer* opponent = new DefensivePlayer("Gerard Pique", 28, 5.8, 0.1);
	forward->TryDribblePast(opponent);

	forward->setGoalCount(42);
	cout << "Payment for this year : " << forward->CalculatePaymentForThisYear()
<< " million EUR" << endl;
}

int main()
{

	TEST_GoalKeeper();
	TEST_DefensivePlayer();
	TEST_MiddleFielderPlayer();
	TEST_ForwardPlayer();
	return 0;
}
```

Problem-Solving Tips

1- Use given UML Diagrams and Test code as guide
2- Payment for a player can be calculated as following:

$$Payment_{GoolKeeper} = paymentPerYear$$

$$Payment_{DefensivePlayer} = paymentPerYear + paymentPerMatch * playedMatchCount$$

$$Payment_{MidFielderPlayer} = paymentPerYear + paymentPerAssist * assistCount$$

$$Payment_{ForwardPlayer} = paymentPerYear + paymentPerGoal * goalCount$$