C++ Loaded Dice OOP

Contents

Time required: 120 minutes

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Pseudocode

1. Write pseudocode or TODO for the exercise

Tutorial 1: OOP Header Files Example

This is an example of how to separate an OOP program into separate files.

```cpp
// main.cpp - Main program file
// Include the header file for class declaration
#include "MyClass.h"
#include <iostream>

int main()
{
    // Create an object of MyClass
    MyClass my_object;

    // Setting private variable using setter method
    my_object.setPrivateVariable(10);

    // Displaying private variable using getter method
    std::cout << my_object.toString() << std::endl;

    return 0;
}
```

Header file declares the class.

```h
// MyClass.h - Header file
// Include string header for std::string
#include <string>
// Include guard to prevent multiple inclusion of the header file
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass
{
private:
    // Private member variable
    int m_PrivateVariable;

public:
    // Constructor declaration
    MyClass();

    // Setter method declaration
    void setPrivateVariable(int value);

    // Getter method declaration
    int getPrivateVariable();

    // Method to return a string representation of the private member variable
    std::string toString();
};

#endif // End of include guard
```

Class implementation show three different types of methods common in OOP.

1. Getter
2. Setter
3. Void display

```cpp
// MyClass.cpp - Implementation file
// Return a string representation of the private member variable
// include the <sstream> header file.
#include <sstream>
#include <iostream>
// Include the header file for class declaration
#include "MyClass.h"

// Definition of the constructor
MyClass::MyClass()
{
    // Initialize private member variable to 0
    m_PrivateVariable = 0;
}

// Definition of the setter method
void MyClass::setPrivateVariable(int value)
{
    // Set the private member variable to the provided value
    m_PrivateVariable = value;
}

// Definition of the getter method
int MyClass::getPrivateVariable()
{
    // Return the private member variable value
    return m_PrivateVariable;
}

// Definition of the method to display a message
std::string MyClass::toString()
{
    // Create a string stream to build the string representation
    std::ostringstream oss;
    oss << "Private member variable value: " << m_PrivateVariable;
    return oss.str();
}
```

In the same folder as this program, create a batch file named:

compile.bat

```
g++ -o main.exe main.cpp MyClass.cpp
pause
```

Double click compile.bat to compile the program into main.exe

Example run:

```
Private member variable value: 10
```

Part 1: Random Dice

Dice are used in many games. One die can be thrown to randomly show a value from 1 through 6.

Create a C++ file named **Die.cpp**

The **Die** class rolls and returns one random int dice value. There isn't any other functionality in the **Die** class.

1. Design a C++ **Die** class with a data field for an integer value.
2. Create a default constructor with no parameters.
3. Create the proper header (.h) and implementation file (.cpp).
4. Create a **rollDie()** method. This method generates a random integer between 1 and 6 and assigns it to a member variable.
5. The **rollDie()** method returns the value generated.

This example assumes you have assigned appropriate values to the static constants.

```
#include <stdlib.h> /* srand, rand */
#include <time.h>   /* time */

// This code goes into the constructor
// rand gives the same sequence each time the program runs
// Initialize random number generator with different values
// time(0) Time since January 1st, 1070 at 00:00:00 AM in milliseconds
srand(time(0));

// This goes into the rollDie method
roll = (rand() % 6) + 1;
```

Part 2: Main Program

Build and test the main program one step at a time.

1. Create two die objects.
2. Roll and display a single die roll from each die object.
3. Compare and display the roll (returned int) from each die object to determine which die won.
4. Comment out the previous display lines. They were only used for debugging.
5. Create a loop to roll the dice 1000 times.
    a. Accumulate the number of times the first **Die** object has a higher value than the second **Die object**.
    b. Display the results.

Compile the program. Create a batch file named **compile_dice_main.bat**

```
g++ -o dice_main.exe dice_main.cpp die.cpp
pause
```

Example run:

```
With two regular die, the first die won: 418 out of 1000
```

Part 3: Loaded Dice

Create a **LoadedDie** class that can give a player a slight advantage over the computer.

1. Copy the Die class header and implementation files. Rename them.
    a. For an extra challenge: use inheritance.
2. A **LoadedDie** never rolls a 1; it only rolls values 2 through 6.

Part 4: Main Program

1. Create an application that rolls two **Die** objects against each other 1,000 times.
    a. Accumulate the number of times the first **Die** object has a higher value than the second **Die object**.
    b. Roll a **Die** object against a **LoadedDie** object 1,000 times and count the number of times the first **Die** wins.
    c. Display the results of each as shown.

2. Compile the program. Create a batch file named **compile_loaded_dice_main.bat**

```
g++ -o loaded_dice_main.exe loaded_dice_main.cpp die.cpp loaded_die.cpp
pause
```

Example application run:

```
With two regular die, the first die won: 418 out of 1000
With one loaded and one regular, the first die won: 507 out of 1000
```

```
With two regular die, the first die won: 410 out of 1000
With one loaded and one regular, the first die won: 479 out of 1000
```

Each run will be different. They will be relatively close to the example.

Assignment Submission

1. Attach the pseudocode.
2. Attach the program files.
3. Attach screenshots showing the successful operation of the program.
4. Submit in Blackboard.