**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **24 June 2020** | **Name:** | **Kavya M M** |
| **Course:** | **C++ Programming** | **USN:** | **4AL17EC040** |
| **Topic:** | **Module 5: Classes and objects**  **Module 6: more on classes** | **Semester & Section:** | **6TH SEM & ‘A’ SEC** |
| **Github Repository:** | **Kavya\_ECE040** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**    **C:\Users\cw\Desktop\24 j1.png**  **C:\Users\cw\Desktop\24 j2.png** |
| What is an Object  Object Oriented Programming is a programming style that is intended to make thinking about programming closer to thinking about the real world.  In programming, objects are independent units, and each has its own identity, just as objects in the real world do.  Objects  An object might contain other objects but they're still different objects.  Objects also have characteristics that are used to describe them. For example, a car can be red or blue, a mug can be full or empty, and so on. These characteristics are also called attributes. An attribute describes the current state of an object.  In programming, an object is self-contained, with its own identity. It is separate from other objects. Each object has its own attributes, which describe its current state. Each exhibits its own behavior, which demonstrates what they can do.https://api.sololearn.com/DownloadFile?id=2427 In computing, objects aren't always representative of physical items. For example, a programming object can represent a date, a time, a bank account. A bank account is not tangible; you can't see it or touch it, but it's still a well-defined object - it has its own identity, attributes, and behavior.  Classes  Objects are created using classes, which are actually the focal point of OOP.  The class describes what the object will be, but is separate from the object itself. In other words, a class can be described as an object's blueprint, description, or definition. You can use the same class as a blueprint for creating multiple different objects. For example, in preparation to creating a new building, the architect creates a blueprint, which is used as a basis for actually building the structure. That same blueprint can be used to create multiple buildings.  Programming works in the same fashion. We first define a class, which becomes the blueprint for creating objects.  Each class has a name, and describes attributes and behavior.  In programming, the term type is used to refer to a class name: We're creating an object of a particular type.  Methods  Method is another term for a class' behavior. A method is basically a function that belongs to a class.  Declaring a Class  Begin your class definition with the keyword class. Follow the keyword with the class name and the class body, enclosed in a set of curly braces. The following code declares a class called BankAccount:  class BankAccount {  };  Define all attributes and behavior (or members) in the body of the class, within curly braces. You can also define an access specifier for members of the class. A member that has been defined using the public keyword can be accessed from outside the class, as long as it's anywhere within the scope of the class object.  Abstraction  Data abstraction is the concept of providing only essential information to the outside world. It's a process of representing essential features without including implementation details.  A good real-world example is a *book*: When you hear the term book, you don't know the exact specifics, i.e.: the page count, the color, the size, but you understand the idea of a book - the abstraction of the book.  Abstraction allows us to write a single bank account class, and then create different objects based on the class, for individual bank accounts, rather than creating a separate class for each bank account.https://api.sololearn.com/DownloadFile?id=2464  Encapsulation  Part of the meaning of the word encapsulation is the idea of "surrounding" an entity, not just to keep what's inside together, but also to protect it. In object orientation, encapsulation means more than simply combining attributes and behavior together within a class; it also means restricting access to the inner workings of that class.  The key principle here is that an object only reveals what the other application components require to effectively run the application. All else is kept out of view.  Access Specifiers  Access specifiers are used to set access levels to particular members of the class. The three levels of access specifiers are public, protected, and private.  A public member is accessible from outside the class, and anywhere within the scope of the class object.  For example:  #include <iostream> #include <string> using namespace std;  class myClass { public: string name; };  int main() { myClass myObj; myObj.name = "SoloLearn"; cout << myObj.name; return 0; }  Private  A private member cannot be accessed, or even viewed, from outside the class; it can be accessed only from within the class. A public member function may be used to access the private members. For example:  #include <iostream> #include <string> using namespace std;  class myClass { public: void setName(string x) { name = x; } private: string name; }; int main() { myClass myObj; myObj.setName("John"); return 0; }Access Specifiers We can add another public method in order to get the value of the attribute.class myClass { public: void setName(string x) { name = x; } string getName() { return name; } private: string name; }; Constructors  Class constructors are special member functions of a class. They are executed whenever new objects are created within that class.  The constructor's name is identical to that of the class. It has no return type, not even void.  For example:  class myClass { public: myClass() { cout <<"Hey"; } void setName(string x) { name = x; } string getName() { return name; } private: string name; };  int main() { myClass myObj;  return 0; }  Constructors can be very useful for setting initial values for certain member variables. A default constructor has no parameters. However, when needed, parameters can be added to a constructor. This makes it possible to assign an initial value to an object when it's created, as shown in the following example:  class myClass { public: myClass(string nm) { setName(nm); } void setName(string x) { name = x; } string getName() { return name; } private: string name; };   |  | | --- | | Source & Header  The header file (.h) holds the function declarations (prototypes) and variable declarations. It currently includes a template for our new MyClass class, with one default constructor. MyClass.h#ifndef MYCLASS\_H #define MYCLASS\_H  class MyClass { public: MyClass(); protected: private: };  #endif // MYCLASS\_H The implementation of the class and its methods go into the source file (.cpp). Currently it includes just an empty constructor. MyClass.cpp#include "MyClass.h"  MyClass::MyClass() { //ctor }    Scope Resolution Operator  The double colon in the source file (.cpp) is called the scope resolution operator, and it's used for the constructor definition:#include "MyClass.h"  MyClass::MyClass() { //ctor } The scope resolution operator is used to define a particular class' member functions, which have already been declared. Remember that we defined the constructor prototype in the header file.  Destructors  Remember constructors? They're special member functions that are automatically called when an object is created. Destructors are special functions, as well. They're called when an object is destroyed or deleted.  #ifndef & #define  We created separate header and source files for our class, which resulted in this header file.#ifndef MYCLASS\_H #define MYCLASS\_H  class MyClass { public: MyClass(); protected: private: };  #endif // MYCLASS\_H  ifndef stands for "if not defined". The first pair of statements tells the program to define the MyClass header file if it has not been defined already. endif ends the condition.  Member Functions  Let's create a sample function called myPrint() in our class. MyClass.hclass MyClass { public: MyClass(); void myPrint(); }; MyClass.cpp#include "MyClass.h" #include <iostream> using namespace std;  MyClass::MyClass() { }  void MyClass::myPrint() { cout <<"Hello"<<endl; }  Constants  A constant is an expression with a fixed value. It cannot be changed while the program is running. Use the const keyword to define a constant variable.  const int x = 42;  Member Initializers  Recall that constants are variables that cannot be changed, and that all const variables must be initialized at time of creation.  C++ provides a handy syntax for initializing members of the class called the member initializer list (also called a constructor initializer).  Composition  In the real world, complex objects are typically built using smaller, simpler objects. For example, a car is assembled using a metal frame, an engine, tires, and a large number of other parts. This process is called composition.  In C++, object composition involves using classes as member variables in other classes. This sample program demonstrates composition in action. It contains Person and Birthday classes, and each Person will have a Birthday object as its member. Birthday:class Birthday { public: Birthday(int m, int d, int y) : month(m), day(d), year(y) {  } private: int month; int day; int year; };  Friend Functions  Normally, private members of a class cannot be accessed from outside of that class. However, declaring a non-member function as a friend of a class allows it to access the class' private members. This is accomplished by including a declaration of this external function within the class, and preceding it with the keyword friend. In the example below, someFunc(), which is not a member function of the class, is a friend of MyClass and can access its private members.class MyClass { public: MyClass() { regVar = 0; } private: int regVar;  friend void someFunc(MyClass &obj); };  This  Every object in C++ has access to its own address through an important pointer called the this pointer. Inside a member function this may be used to refer to the invoking object. Let's create a sample class:class MyClass { public: MyClass(int a) : var(a) { } private: int var; };  Operator Overloading  Most of the C++ built-in operators can be redefined or overloaded. Thus, operators can be used with user-defined types as well (for example, allowing you to add two objects together).  This chart shows the operators that can be overloaded.https://api.sololearn.com/DownloadFile?id=2463 | |