**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **24-06-2020** | **Name:** | **Karthik J** |
| **Course:** | C++ Tutorials - Solo Learn | **USN:** | **4AL16EC030** |
| **Topic:** | Classes and Objects | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image Section**   **C++ Classes/Objects** C++ is an object-oriented programming language.  Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.  Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "class members".  A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects. Create a Class To create a class, use the class keyword: Example Create a class called "MyClass":  class MyClass {       // The class   public:             // Access specifier     int myNum;        // Attribute (int variable)     string myString;  // Attribute (string variable) }; Create an Object In C++, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.  To create an object of MyClass, specify the class name, followed by the object name.  To access the class attributes (myNum and myString), use the dot syntax (.) on the object: Example Create an object called "myObj" and access the attributes:  class MyClass {       // The class   public:             // Access specifier     int myNum;        // Attribute (int variable)     string myString;  // Attribute (string variable) };  int main() {   MyClass **myObj**;  // Create an object of MyClass    // Access attributes and set values   **myObj.myNum** = 15;    **myObj.myString** = "Some text";    // Print attribute values   cout << myObj.myNum << "\n";    cout << myObj.myString;    return 0; } Multiple Objects You can create multiple objects of one class: Example // Create a Car class with some attributes class Car {   public:     string brand;        string model;     int year; };  int main() {   // Create an object of Car   Car carObj1;   carObj1.brand = "BMW";   carObj1.model = "X5";   carObj1.year = 1999;    // Create another object of Car   Car carObj2;   carObj2.brand = "Ford";   carObj2.model = "Mustang";   carObj2.year = 1969;    // Print attribute values   cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";   cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";   return 0;  } **Class Methods** Methods are **functions** that belongs to the class.  There are two ways to define functions that belongs to a class:   * Inside class definition * Outside class definition   In the following example, we define a function inside the class, and we name it "myMethod".  **Note:** You access methods just like you access attributes; by creating an object of the class and by using the dot syntax (.): Inside Example class MyClass {        // The class   public:              // Access specifier     void myMethod() {  // Method/function defined inside the class       cout << "Hello World!";     } };  int main() {   MyClass myObj;     // Create an object of MyClass   myObj.myMethod();  // Call the method   return 0; }  To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifiying the name of the class, followed the scope resolution :: operator, followed by the name of the function: Outside Example class MyClass {        // The class   public:              // Access specifier     void myMethod();   // Method/function declaration };  // Method/function definition outside the class void **MyClass::myMethod()** {   cout << "Hello World!"; }  int main() {   MyClass myObj;     // Create an object of MyClass   myObj.myMethod();  // Call the method   return 0; } Parameters You can also add parameters:  Example #include <iostream> using namespace std;  class Car {   public:     int speed(int maxSpeed); };  int Car::speed(int maxSpeed) {   return maxSpeed; }  int main() {   Car myObj; // Create an object of Car   cout << myObj.speed(200); // Call the method with an argument   return 0; } Constructors A constructor in C++ is a **special method** that is automatically called when an object of a class is created.  To create a constructor, use the same name as the class, followed by parentheses (): Example class MyClass {     // The class   public:           // Access specifier     MyClass() {     // Constructor       cout << "Hello World!";     } };  int main() {   MyClass myObj;    // Create an object of MyClass (this will call the constructor)   return 0; }  **Note:** The constructor has the same name as the class, it is always public, and it does not have any return value. **Constructor Parameters** Constructors can also take parameters (just like regular functions), which can be useful for setting initial values for attributes.  The following class have brand, model and year attributes, and a constructor with different parameters. Inside the constructor we set the attributes equal to the constructor parameters (brand=x, etc). When we call the constructor (by creating an object of the class), we pass parameters to the constructor, which will set the value of the corresponding attributes to the same: Example class Car {        // The class   public:          // Access specifier     string brand;  // Attribute     string model;  // Attribute     int year;      // Attribute     Car(string x, string y, int z) { // Constructor with parameters       brand = x;       model = y;       year = z;     } };  int main() {   // Create Car objects and call the constructor with different values   Car carObj1("BMW", "X5", 1999);   Car carObj2("Ford", "Mustang", 1969);    // Print values   cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";   cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";   return 0; }  Just like functions, constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution :: operator, followed by the name of the constructor (which is the same as the class): Example class Car {        // The class   public:          // Access specifier     string brand;  // Attribute     string model;  // Attribute     int year;      // Attribute     Car(string x, string y, int z); // Constructor declaration };  // Constructor definition outside the class Car::Car(string x, string y, int z) {   brand = x;   model = y;   year = z; }  int main() {   // Create Car objects and call the constructor with different values   Car carObj1("BMW", "X5", 1999);   Car carObj2("Ford", "Mustang", 1969);    // Print values   cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";   cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";   return 0; } **Access Specifiers** By now, you are quite familiar with the public keyword that appears in all of our class examples: Example class MyClass {  // The class   **public:**        // Access specifier     // class members goes here };  The public keyword is an **access specifier.** Access specifiers define how the members (attributes and methods) of a class can be accessed. In the example above, the members are public - which means that they can be accessed and modified from outside the code.  However, what if we want members to be private and hidden from the outside world?  In C++, there are three access specifiers:   * public - members are accessible from outside the class * private - members cannot be accessed (or viewed) from outside the class * protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about [Inheritance](https://www.w3schools.com/cpp/cpp_inheritance.asp) later.   In the following example, we demonstrate the differences between public and private members: Example class MyClass {   **public:**    // Public access specifier     int x;   // Public attribute   **private:**   // Private access specifier     int y;   // Private attribute };  int main() {   MyClass myObj;   myObj.x = 25;  // Allowed (public)   myObj.y = 50;  // Not allowed (private)   return 0; }  If you try to access a private member, an error occurs:  error: y is private  **Note:** It is possible to access private members of a class using a public method inside the same class. See the next chapter ([Encapsulation](https://www.w3schools.com/cpp/cpp_encapsulation.asp)) on how to do this.  **Tip:** It is considered good practice to declare your class attributes as private (as often as you can). This will reduce the possibility of yourself (or others) to mess up the code. This is also the main ingredient of the [Encapsulation](https://www.w3schools.com/cpp/cpp_encapsulation.asp) concept, which you will learn more about in the next chapter.  **Note:** By default, all members of a class are private if you don't specify an access specifier: Example class MyClass {   int x;   // Private attribute   int y;   // Private attribute }; **Encapsulation** The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class). If you want others to read or modify the value of a private member, you can provide public **get** and **set** methods. Access Private Members To access a private attribute, use public "get" and "set" methods: Example #include <iostream> using namespace std;  class Employee {   private:     // Private attribute     int salary;    public:     // Setter     void setSalary(int s) {       salary = s;     }     // Getter     int getSalary() {       return salary;     } };  int main() {   Employee myObj;   myObj.setSalary(50000);   cout << myObj.getSalary();   return 0; } Example explained The salary attribute is private, which have restricted access.  The public setSalary() method takes a parameter (s) and assigns it to the salary attribute (salary = s).  The public getSalary() method returns the value of the private salary attribute.  Inside main(), we create an object of the Employee class. Now we can use the setSalary() method to set the value of the private attribute to 50000. Then we call the getSalary() method on the object to return the value. |