**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **24/06/2020** | **Name:** | **Lavanya B** |
| **Course:** | **C++ programming** | **USN:** | **4al17ec043** |
| **Topic:** | **Classes and objects** | **Semester & Section:** | **6th A** |
| **Github Repository:** | **Lavanya-B** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **Objects**  **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 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.**  **Objects can have multiple attributes (the mug can be empty, red and large).**  **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.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class BankAccount {**  **public:**  **void sayHi() {**  **cout << "Hi" << endl;**  **}**  **};**  **int main()**  **{**  **BankAccount test;**  **test.sayHi();**  **}**  **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.**  **Abstraction means, that we can have an idea or a concept that is completely separate from any specific instance.**  **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.**  **This allows us to change attributes and implementation of methods without altering the overall program.**  **Eg:**  **#include <iostream>**  **#include <string>**  **using namespace std;**  **class myClass {**  **public:**  **string name;**  **};**  **int main() {**  **myClass myObj;**  **myObj.name = "SoloLearn";**  **cout << myObj.name;**  **return 0;**  **}**  **Constructor**  **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.**  **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.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class myClass {**  **public:**  **myClass(string nm) {**  **setName(nm);**  **}**  **void setName(string x) {**  **name = x;**  **}**  **string getName() {**  **return name;**  **}**  **private:**  **string name;**  **};**  **int main() {**  **myClass ob1("David");**  **myClass ob2("Amy");**  **cout << ob1.getName();**  **}**  **Destructors**  **Destructors are special functions, as well. They're called when an object is destroyed or deleted.**  **The name of a destructor will be exactly the same as the class, only prefixed with a tilde (~). A destructor can't return a value or take any parameters.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class MyClass**  **{**  **public:**  **MyClass();**  **~MyClass();**  **};**  **MyClass::MyClass()**  **{**  **cout<<"Constructor"<<endl;**  **}**  **MyClass::~MyClass()**  **{**  **cout<<"Destructor"<<endl;**  **}**  **int main() {**  **MyClass obj;**  **}**  **Friend function**  **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.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class MyClass {**  **public:**  **MyClass() {**  **regVar = 0;**  **}**  **private:**  **int regVar;**  **friend void someFunc(MyClass &obj);**  **};**  **void someFunc(MyClass &obj) {**  **obj.regVar = 42;**  **cout << obj.regVar;**  **}**  **int main() {**  **MyClass obj;**  **someFunc(obj);**  **}** |