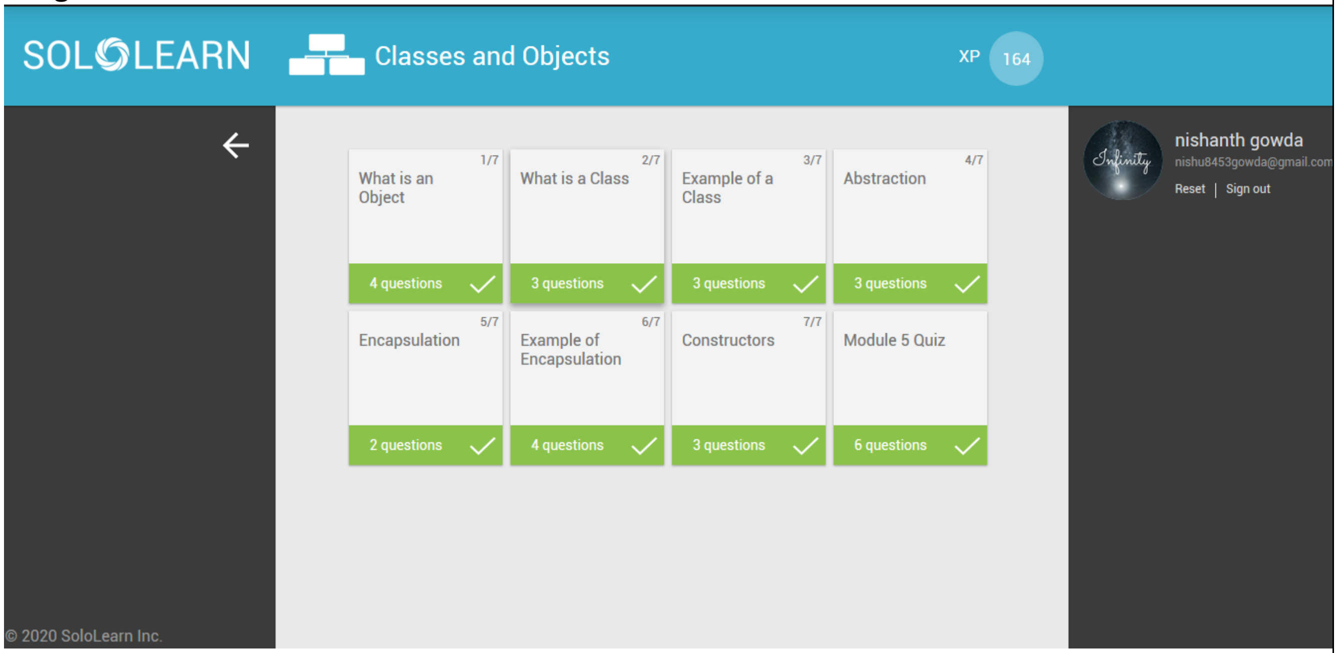
DAILY ASSESSMENT FORMAT

Date:	25/June/2020	Name:	nishanth
Course:	C++ programming	USN:	4al17ec063
Topic:	Class and objects	Semester & Section:	6th b
GitHub Repository:	nishanthvr		

FORENOON SESSION DETAILS

Image of session


SOLOLEARN
Classes and Objects
XP 164
nishanth gowda
nishu8453gowda@gmail.com
Reset | Sign out
1/7 What is an Object 4 questions
2/7 What is a Class 3 questions
3/7 Example of a Class 3 questions
4/7 Abstraction 3 questions
5/7 Encapsulation 2 questions
6/7 Example of Encapsulation 4 questions
7/7 Constructors 3 questions
Module 5 Quiz 6 questions
© 2020 SoloLearn Inc.

What is an Object

Object **O**riented **P**rogramming 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.
An apple is an object; so is a mug. Each has its unique **identity**. It's possible to have two mugs that look identical, but they are still separate, unique objects.

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.
Objects can have multiple attributes (the mug can be **empty**, **red** and **large**).
An object's **state** is independent of its type; a cup might be full of water, another might be empty.

What is a Class

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**.
Attributes are also referred to as **properties** or **data**.

A Class Example

For example, if we are creating a banking program, we can give our class the following characteristics:
name: BankAccount
attributes: accountNumber, balance, dateOpened
behavior: open(), close(), deposit()

The class specifies that each object should have the defined attributes and behavior. However, it doesn't specify what the actual data is; it only provides a **definition**.

Once we've written the class, we can move on to create objects that are based on that class.
Each object is called an **instance** of a class. The process of creating objects is called **instantiation**.
Each object has its own identity, data, and behavior.

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.
This is called **data hiding**
For example, if we take our **BankAccount** class, we do not want some other part of our program to reach in and change the **balance** of any object, without going through the **deposit()** or **withdraw()** behaviors.
We should **hide** that attribute, control access to it, so it is accessible only by the object itself.

This way, the **balance** cannot be directly changed from outside of the object and is accessible only using its methods.
This is also known as "**black boxing**", which refers to closing the inner working zones of the object, except of the pieces that we want to make public.
This allows us to change attributes and implementation of methods without altering the overall program. For example, we can come back later and change the data type of the **balance** attribute.
In summary the benefits of encapsulation are:
- **Control** the way data is accessed or modified.
- Code is more **flexible** and easy to change with new requirements.
- **Change** one part of code without affecting other part of code.

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;
}
```

//Outputs "Hey"Try It Yourself

Now, upon the creation of an object of type **myClass**, the constructor is automatically called.