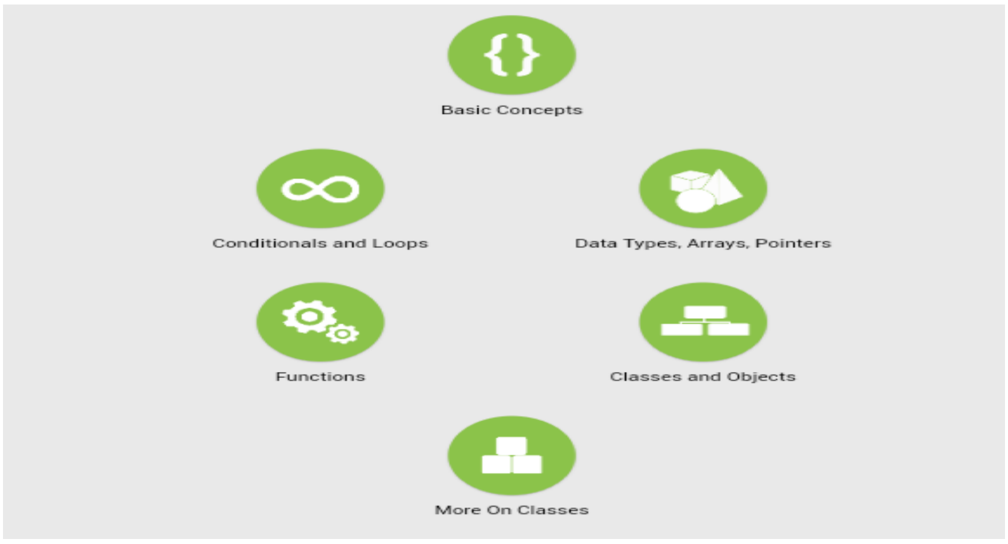
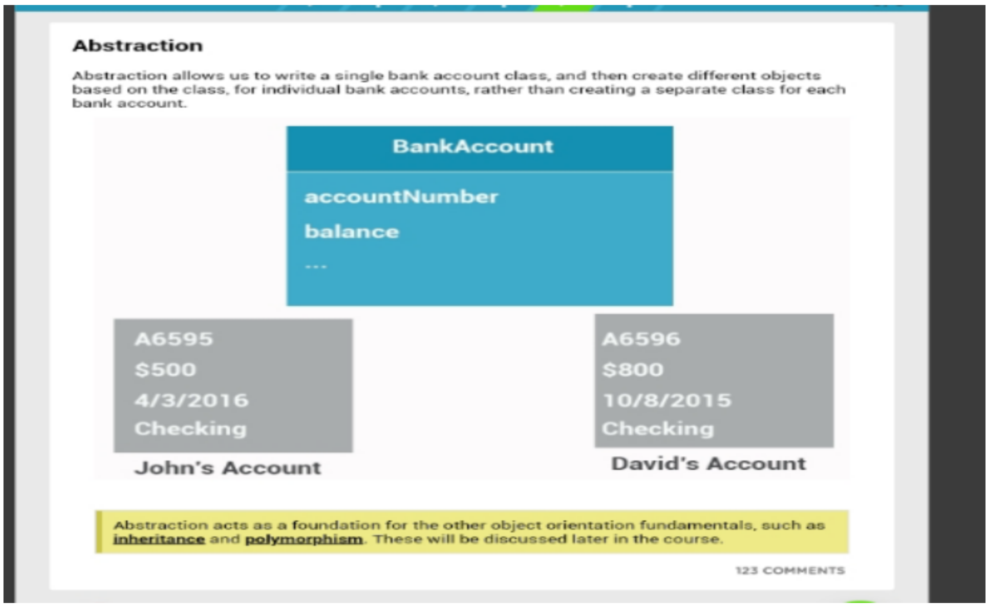
DAILY ASSESSMENT FORMAT

Date:	**24/06/2020**	**Name:**	Nichenametla Bhargavi
Course:	Programming in C++	**USN:**	**4AL17EC061**
Topic:	**Classes and Objects**	**Semester & Section:**	**6th Sem A Sec**
Github Repository:	Bhargavi_Nichenametla		

SESSION

Image of the session





Abstraction

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.

Abstraction acts as a foundation for the other object orientation fundamentals, such as **inheritance** and **polymorphism**. These will be discussed later in the course.

123 COMMENTS

Report – Report can be typed or hand written for up to two pages.

Methods:

Method is another term for a class' behavior. A method is basically a function that belongs to a class.

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, itdoesn'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.

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 {

};
```

Declaring a Class

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.

Creating a Class:
Let's create a class with one public method, and have it print out "Hi".

```
class BankAccount {
public:
 void sayHi() {
 cout << "Hi" << endl;
 }

};
```

The next step is to instantiate an object of our BankAccount class, in the same way we define variables of a type, the difference being that our object's type will be BankAccount.

```
int main()
{
BankAccount test;
 test.sayHi();
}
```

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.

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.

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.

Source & Header:

To use our classes in our main, we need to include the header file.
For example, to use our newly created MyClass in main:

```
#include <iostream>
#include "MyClass.h"
using namespace std;
int main() {
MyClass obj;
}
```

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.
* 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.

```
class MyClass {
 public:
 ~MyClass() {
 // some code

 }

};
```

After declaring the destructor in the header file, we can write the implementation in the source file MyClass.cpp:

```
#include "MyClass.h"
#include <iostream>
using namespace std;
MyClass::MyClass()
{
 cout<<"Constructor"<<endl;
}
MyClass::~MyClass()
{

 cout<<"Destructor"<<endl;

}
```

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.

"This" Keyword Usage:

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;

};
```