Object Oriented Programming

Tushar B. Kute,
http://tusharkute.com

Data Abstraction

- Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
- Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

Data Abstraction

- Let's take one real life example of a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, BUT you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.
- Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having any knowledge of its internals.

Data Abstraction

- In C++, classes provides great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.
- For example, your program can make a call to the sort() function without knowing what algorithm the function actually uses to sort the given values.
- In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

Data Abstraction

- In C++, we use classes to define our own abstract data types (ADT). You can use the cout object of class ostream to stream data to standard output like this –

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello C++" <<endl;
  return 0;
}
```

- Here, you don't need to understand how cout displays the text on the user's screen. You need to only know the public interface and the underlying implementation of ‘cout’ is free to change.

Data Abstraction

- In C++, we use access labels to define the abstract interface to the class. A class may contain zero or more access labels –
  - Members defined with a public label are accessible to all parts of the program. The data-abstraction view of a type is defined by its public members.
  - Members defined with a private label are not accessible to code that uses the class. The private sections hide the implementation from code that uses the type.

Data Abstraction

- Data abstraction provides two important advantages –
  - Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.
  - The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

Data Abstraction

- Example.

Data Abstraction

- Abstraction separates code into interface and implementation.
- So while designing your component, you must keep interface independent of the implementation so that if you change underlying implementation then interface would remain intact.
- In this case whatever programs are using these interfaces, they would not be impacted and would just need a recompilation with the latest implementation.

Polymorphism

- The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Overloading

- C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.
- An overloaded declaration is a declaration that is declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

Overloading

- When you call an overloaded function or operator, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions.
- The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Overloading

- Example.

Operators Overloading

- You can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.
- Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

```
Box operator+(const Box&);
```

- declares the addition operator that can be used to add two Box objects and returns final Box object.

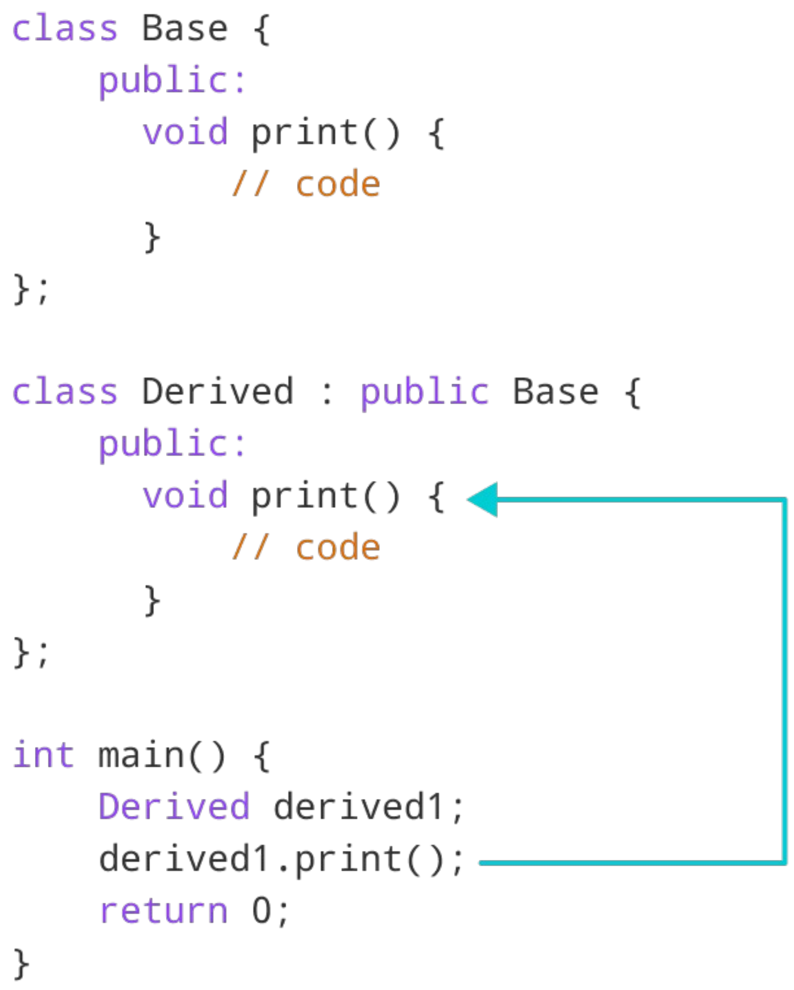
Operators Overloading

- Example.

Function Overriding

- Suppose, the same function is defined in both the derived class and the based class.
- Now if we call this function using the object of the derived class, the function of the derived class is executed.
- This is known as function overriding in C++. The function in derived class overrides the function in base class.

Function Overriding



```
class Base {
    public:
      void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
      }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

Function Overriding

- Example.

Thank you

@mitu_skillologies	@mITuSkillologies	@mitu_group	@mitu-skillologies	@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com