Object Oriented Programming

Polymorphism (recap), Virtual destructor, Abstraction, Encapsulation, Abstract class and Concrete class

Recap (Polymorphism)

- Polymorphism types
  - Overloading
  - Overriding
- Binding types
  - Static
  - Dynamic

Virtual destructor

- Deleting a derived class object using a pointer to a base class that has a non-virtual destructor results in undefined behavior. To correct this situation, the base class should be defined with a virtual destructor. For example, following program results in undefined behavior

```
//without virtual destructor

class base {
public:
    base()
    {
     cout<<"Constructing base \n";
     }
     ~base()
    {
     cout<<"Destructing base \n";
    }
};
```

```
class derived: public base {
public:
            derived()
            {
            cout<<"Constructing derived \n";
             }
            ~derived()
            {
            cout<<"Destructing derived \n";
            }
};
int main()
{
derived d;;
base *b = &d;

return 0;
}
```

```
Output:

Constructing base
Constructing derived
Destructing base
```

- Making base class destructor virtual guarantees that the object of derived class is destructed properly, i.e., both base class and derived class destructors are called.

```
//with virtual destructor

class base {
public:
    base()
    {
     cout<<"Constructing base \n";
     }
    virtual ~base()
    {
     cout<<"Destructing base \n";
    }
};
```

```
class derived: public base {
public:
        derived()
        {
        cout<<"Constructing derived \n";
         }
        ~derived()
        {
        cout<<"Destructing derived \n";
        }
};
int main()
{
derived *d = new derived();
base *b = d;
delete b;
return 0;
}
```

Output:

```
Constructing base
Constructing derived
Destructing derived
Destructing base
```

Abstract class – pure virtual function

An abstract class is a class that is designed to be specifically used as a base class. An abstract class contains at least one **pure virtual function.**

You declare a pure virtual function by using a *pure specifier* (= 0) in the declaration of a virtual member function in class.

Syntax (Abstract class - Pure virtual function)

```
class AB {
public:
  virtual void f() = 0;
};
```

Limitations of Abstract class

- You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class. You can, however, declare pointers and references to an abstract class.

Examples - Limitations

```
class A {  virtual void f() = 0;};
class B : A {  virtual void f() { }};
A g();
void h(A);
int main() {
A a;
A* pa;
B b;
}
```

- Class A is an abstract class. The compiler would not allow the function declarations A g() or void h(A).

Abstract class facts

- A class is abstract if it has at least one pure virtual function.
- We cannot instantiate an abstract class type; however, we can have pointers and references of abstract class type.
- If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.
- An abstract class can have constructors.

```
//Abstract class example program

class Base
{
int x;
public:
        virtual void fun() = 0;
        int getX() { return x; }
};
```

```
// This class inherits from Base and
//implements fun()
class Derived: public Base
{
        int y;
public:
        void fun() { cout << "fun() called"; }
};

int main()
{
        Derived d;
        d.fun();
        return 0;
}
```

```
//Instantiating abstract class ( cause error)
class Test
{
int x;
public:
        virtual void show() = 0;
        int getX() { return x; }
};
int main()
{
        Test t;
        return 0;
}
```

```
//Declaring pointer to abstract class
class Base
{
public:
        virtual void show() = 0;
};
class Derived: public Base
{
public:
        void show() { cout << "In Derived \n"; }
};
int main()
{       Base *bp = new Derived();
        bp->show();
        return 0;
}
```

```
// An abstract class with constructor
class Base
{
protected:
int x;
public:
virtual void fun() = 0;
Base(int i) { x = i; }  };
class Derived: public Base
{           int y;
public:
            Derived(int i, int j):Base(i) { y = j; }
            void fun() { cout << "x = " << x << ", y = " << y; }
};
int main()
{           Derived d(4, 5);
            d.fun();
            return 0; }
```

```
//Not overriding pure virtual function in derived class
class Base
{
public:
        virtual void show() = 0;
};

class Derived : public Base { };

int main(void)
{
Derived d;
return 0;
}
```

```
//Not overriding pure virtual function in derived class
class AB {
public: virtual void f() = 0;
};
class D2 : public AB {
void g();
};
int main() {
D2 d;
}
```

The compiler will not allow the declaration of object `d` because `D2` is an abstract class; it inherited the pure virtual function `f()` from `AB`.
The compiler will allow the declaration of object `d` if you define function `D2::f()`, as this overrides the inherited pure virtual function `AB::f()`. Function `AB::f()` needs to be overridden if you want to avoid the abstraction of `D2`.

Concrete class

- A concrete class is an ordinary class which has no purely virtual functions and hence can be instantiated.

- It has an implementation for all of its methods. There cannot have any unimplemented methods.

Abstraction

- Abstraction is one of the features of Object Oriented Programming, where you show only relevant details to the user and hide irrelevant details. For example, when you send an email to someone you just click send and you get the success message, what actually happens when you click send, how data is transmitted over network to the recipient is hidden from you (because it is irrelevant to you).

Advantages of abstraction

- The major advantage of using this feature is that when the code evolves and you need to make some adjustments in the code then you only need to modify the high level class where you have declared the members as private. Since none class is accessing these data members directly, you do not need to change the low level(user level) class code.
  Imagine if you had made these data members public, if at some point you want to change the code, you would have to make the necessary adjustments to all the classes that are accessing the members directly.

Encapsulation

- **Encapsulation** is a process of combining data members and functions in a single unit called class. This is to prevent the access to the data directly, the access to them is provided through the functions of the class. It is one of the popular feature of Object Oriented Programming(OOP) that helps in data hiding.

Data hiding

- **Data hiding** is an object-oriented programming technique of hiding internal object details i.e. data members. Data hiding guarantees restricted data access to class members & maintain object integrity

References

- https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.cbclx01/cplr142.htm
- https://www.geeksforgeeks.org/abstraction-in-c/
- https://www.geeksforgeeks.org/virtual-destructor/
- https://beginnersbook.com/2017/09/abstraction-in-c-with-example/
- https://www.geeksforgeeks.org/pure-virtual-functions-and-abstract-classes/