Data Structures and Object Oriented Programming using C++

Ahsan Ijaz

Polymorphism

- Polymorphism is a generic term that means 'many shapes'. In C++ the simplest form of Polymorphism is overloading of functions.
- in the OOP form of polymorphism, a virtual function in the base class Point and then overriding it in the derived class Circle.

Pointer to Base Class

Definition

A pointer to a derived class is type-compatible with a pointer to its base class.

```
1  BaseClass BaseObj;
2  BaseClass * PBClass = &BaseObj;
3  DerivedClass DerivedObj;
4  BaseClass * pDclass = &DerivedObj;
```

Pointer to Base Class

```
1  #include <iostream>
2  using namespace std;
3
4  class animal
5  {
6  public:
7  void eat();
8  void sleep();
9  void breathe();
10 };
11
12 class fish : public animal
13 {
14 public:
15 void breathe(); //same function as derived class
16 }
17
18 void animal::breathe(){cout<<"Breathing..."<<endl;}
19 void fish::breathe(){cout<<"Bubbling..."<<endl;}
```

Main Implementation

```
1  int main()
2  {
3  animal animal_object;
4  fish bigfish;
5  animal *pointer; //pointer of kind animal
6
7  pointer= &animal_object;
8  pointer->breathe();
9  /*pointer of kind animal
10   accessing a fish object*/
11 pointer= &bigfish;
12 pointer->breathe();
13
14 return 0;
15 }
```

Output:

Breathing...
Breathing...

Main Implementation

```
1  int main()
2  {
3  animal animal_object;
4  fish bigfish;
5  animal *pointer; //pointer of kind animal
6
7  pointer= &animal_object;
8  pointer->breathe();
9  /*pointer of kind animal
10   accessing a fish object*/
11 pointer= &bigfish;
12 pointer->breathe();
13
14 return 0;
15 }
```

Output:

Breathing...
Breathing...

Use of Virtual function

```
1  #include <iostream>
2  using namespace std;
3
4  class animal
5  {
6  public:
7  void eat();
8  void sleep();
9  virtual void breathe(); //virtual keyword added
10 };
11
12 class fish : public animal
13 {
14 public:
15 void breathe();
16 }
17
18 void animal::breathe(){cout<<"Breathing..."<<endl;}
19 void fish::breathe(){cout<<"Bubbling..."<<endl;}
```

Main Implementation

```
1  int main()
2  {
3  animal animal_object;
4  fish bigfish;
5  animal *pointer;
6
7  pointer= &animal_object;
8  pointer->breathe();
9
10 pointer= &bigfish;
11 pointer->breathe(); //Now this calls
12 // function of fish class
13
14 return 0;
15 }
```

Output:

Breathing...
Bubbling...

Main Implementation

```
1  int main()
2  {
3  animal animal_object;
4  fish bigfish;
5  animal *pointer;
6
7  pointer= &animal_object;
8  pointer->breathe();
9
10 pointer= &bigfish;
11 pointer->breathe(); //Now this calls
12 // function of fish class
13
14 return 0;
15 }
```

Output:

Breathing...

Bubbling...

Function Over Ride

Definition

Changing implementation of a function from base class when the function is called through a derived class object.

Function Over Riding Example

```
1  class B_class{
2  public:
3  void functionA(){ /*...*/ }
4  };
5  class D_class : public B_class{
6  public:
7  void functionA(){ /*...*/ }
8  };
9  int main(){
10 D_class x;
11 x.functionA();
12 }
```

Polymorphism-Base Class

In case of pure virtual function, the base class **cannot** be instantiated or created.

```
1  class bird
2  {
3  public:
4     virtual void fly() = 0; // Pure virtual Function
5  };
```

Polymorphism-Derived Classes

```
1   class hummingbird : public bird
2   {
3   public:
4     virtual void fly()
5     {
6       flap_wings_like_crazy();
7     }
8   };
9
10  class albatross : public bird
11  {
12  public:
13    virtual void fly()
14    {
15      glide_majestically_over_the_waves();
16    }
17  };
```

Polymorphism-Implementation

Since **fly()** is virtual function, **migrate()** will call the appropriate **fly()** function.

```
1  void migrate(bird* bird_pointer)
2  {
3      bird_pointer->fly();
4  }
5  int main()
6  {
7  hummingbird h;
8  albatross a;
9
10 migrate(&h);
11 migrate(&a);
12 }
```