Polymorphism

- **P**olymorphism is a concept where a single name may denote objects of different classes that are related by some common base class[Booch].
- In polymorphic languages some values and variables may have more than one type
- A function is polymorphic if it may be applied to arguments of different types

The universal polymorphism

- Polymorphism is constructed by poly(means many) + morph (state, form,etc)
- For example, several people in the class heard a big explosion(getting a message “explosion”), some shouted ”what’s the matter?” some rushed out the classroom for details, and some did not care about it. This is polymorphism!!!

Examples

- Write a program to maintain a list of shapes created by the user, and print the shapes when needed.
- The shapes needed in the application are:
  - points
  - lines
  - rectangles
  - circles
  - etc...

In Conventional Programs

You should write:

```
if (Shape.type is circle) then
  DrawCircle(Shape);
else if (Shape.type is rectangle) then
  DrawRectangle(Shape);
else if (Shape.type is point) then
  DrawPoint(Shape);
else if( Shape.type is line) then
  DrawLine(Shape);
```

Using Polymorphism

- You need only to write:
  Shape.Draw()

Static and Dynamic Binding

- When a reference to a member function is resolved at compile time, then static binding is used.
- When a reference to a member function can only be resolved at run-time, then this is called dynamic binding.

Polymorphism and Dynamic Binding

- To implement polymorphism, a language must support dynamic binding.
  - Polymorphism----- a concept
  - Dynamic binding -----implementation
- Why?
  - With static binding, can you support a polymorphism?

Polymorphism and Dynamic Binding

- Classical paradigm
  - function open_disk_file()
  - function open_tape_file()
  - function open_diskette_file()
- Object-Oriented paradigm
  - Function My_File.open_file()

Polymorphism and Dynamic Binding

- All that is needed is myFile.open()
  - Correct method invoked at run-time(dynamically)
- Method open can be applied to objects of different classes
  - Polymorphism

Negative Aspects

- Negative impact on maintenance
  - Hard to understand if multiple possibilities for specific method
    - Note:Do not use the same name when the methods have little similarities.

Polymorphism in C++

- Virtual functions
- Abstract classes

Polymorphism in C++

- It gives us the ability to manipulate instances of derived class through a set of operations defined in their base class.
- Each derived class can implement the operations differently, while retaining a common class interface provided by the base class.

Polymorphism in C++

One of the greater advantages of deriving classes is that **a pointer to a derived class is type-compatible with a pointer to its base class**. This section is fully dedicated to taking advantage of this powerful C++ feature.

```
// pointers to base class
#include <iostream.h>

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
```

```
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << endl;  (?)
  cout << sqre.area() << endl;
  return 0;
}
```

```
Output:
20
10
```

To make it possible for the pointers to class CPolygon admit area() as a valid member, this should also have been declared in the base class and not only in its derived ones.

Virtual Member Functions

- Virtual Function
  - A non-static member function prefaced by the *virtual* specifier.
  - It tells the compiler to generate code that selects the appropriate version of this function at run time.

```
// virtual members
#include <iostream.h>
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void)
      { return (0); }
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
```

```
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon poly;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  CPolygon * ppoly3 = &poly;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly3->set_values (4,5);
  cout << ppoly1->area() << endl;
  cout << ppoly2->area() << endl;
  cout << ppoly3->area() << endl;
  return 0;
}
```

```
Output:
20
10
0
```

What Happened ...

`area()` has been defined as `virtual` because it is later redefined in derived classes. You can verify if you want that if you remove this word (`virtual`) from the code and then you execute the program the result will be `0` for the three polygons instead of `20,10,0`. That would be because instead of calling to the corresponding `area()` function for each object (`CRectangle::area()`, `CTriangle::area()` and `CPolygon::area()`, respectively), `CPolygon::area()` will be called for all of them since the calls are via a pointer to `CPolygon`.

What Happened …

Therefore what the word `virtual` does is to allow that the member of a derived class with the same name as one in the base class is suitably called when a pointer to the base class is used.

Abstract Classes

- An abstract class is a class that can only be a base class for other classes.
- Abstract classes represent concepts for which objects cannot exist.
- A class that has no instances is an abstract class
- Concrete Classes are used to instantiate objects

Abstract Classes

- In C++, an abstract class either contains or inherits at least one pure virtual function.
- A pure virtual function is a virtual function that contains a pure-specifier, designated by the “=0”.

Example

```
// abstract class CPoligon
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
  };
```

CPolygon poly;

```
CPolygon * ppoly1;
CPolygon * ppoly2;
```

```
// virtual members
#include <iostream.h>

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
```

```
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << ppoly1->area() << endl;
  cout << ppoly2->area() << endl;
  return 0;
}
```

If you review the program you will notice that we can refer to objects of different classes using a unique type of pointer (CPolygon*). This can be tremendously useful. Imagine, now we can create a function member of CPolygon that is able to print on screen the result of the area() function independently of which of the derived classes is.

```
#include <iostream.h>

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
    void printarea (void)
      { cout << this->area() << endl; }    (?)
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
```

```
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```

Abstract classes and virtual members grant to C++ the polymorphic characteristics that make object-oriented programming a so useful instrument.

An Abstract Derived Class

- If in a derived class a pure virtual function is not defined, the derived class is also considered an abstract class.
- When a derived class does not provide an implementation of a virtual function the base class implementation is used.
- It is possible to declare pointer variables to abstract classes.

Inheritance of Java

- Directly support single inheritance only
- Methods can be final (cannot be overriden)

Dynamic Binding of Java

- In Java, all messages are dynamically bound to methods, unless the method is final (means it cannot be overriden; therefore, dynamic binding serves no purpose)