Object Polymorphism

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Polymorphism: Compiletime

Polymorphism is the mechanism of **decoupling the actual behavior** of the **functions from** their **names**.

- **Function Overloading**: The same function name within a class, but with a different signature (different parameters), **compile-time polymorphism**.
- The same function call with different parameters results in:
  - execution of behavior that is specific to the parameter list.
  - possibly different behavior than that of with the other parameter lists.

Polymorphism: Runtime

- In **runtime polymorphism**, the actual **object** referred by a **reference** or a **pointer** is resolved at runtime and then the **function calls** are resolved to the version of the functions associated with that actual object. It is also known as dynamic or late binding polymorphism.

- Runtime polymorphism is achieved through **virtual functions**, **function overriding**, and **reference** or **pointer**.

Polymorphism: Runtime

- Base class **virtual function** (same name and parameter list) is implemented with different behaviors in the derived classes. This feature of derived classes that replaces the behavior of a base class with new or modified behavior is called **function *overriding***.

- The same function call on different types of objects results in execution of behavior that is specific to that particular object.

Polymorphism: Runtime

- A derived class (D) has all the members of its base class (B)
  - Class D is a subtype of Class B.
  - Class D can be used anytime class B is expected.
  - Class D object can be used as a class B variable.
  - **Class D object** can also be used as a **class B reference** or **pointer.**
- If class D overrides some of class B functions and class D object is used with a class B reference or pointer.
  - Class D version of the overrode function will be executed even though they are called through class B reference or pointer.

Polymorphism: Runtime

```
class Polygon {
  protected:
        int width;
        int height;
  public:
    Polygon(int width, int height): width(width), height(height) {}
    virtual int area (){ return 0; }
};
```

Polymorphism: Runtime

```
class Rectangle: public Polygon {
  public:
    Rectangle(int width, int height): Polygon(width, height) {}
    int area () override { return width * height; }
};

class Triangle: public Polygon {
  public:
    Triangle(int width, int height): Polygon(width, height) {}
    int area () override { return width * height / 2; }
};
```

Polymorphism: Runtime

```
int main () {
    Rectangle rect(7,8);
    Triangle trgl(7,8);
    Polygon poly(7,8);
    Polygon* ppoly = &rect;
    cout << ppoly->area() << endl;          //56 = 7x8
    ppoly = &trgl;
    cout << ppoly->area() << endl;          //28 = 7x8/2
    ppoly = &poly;
    cout << ppoly->area() << endl;          //0
    return 0;
}
```

Polymorphism: Abstract Class

```
class Polygon {
  protected:
        int width;
        int height;
  public:
    Polygon(int width, int height): width(width), height(height) {}
    virtual int area () = 0;       //Pure virtual function
};
```

As area() function is pure virtual, all derived classes from Polygon base class should override this function. Otherwise, the derived class will become another abstract class.

Polymorphism: Abstract Class

```
class Rectangle: public Polygon {
  public:
    Rectangle(int width, int height): Polygon(width, height) {}
    int area () override { return width * height; }
};

class Triangle: public Polygon {
  public:
    Triangle(int width, int height): Polygon(width, height) {}
    int area () override { return width * height / 2; }
};
```

Polymorphism: Abstract Class

```
int main () {
    Rectangle rect(7,8);
    Triangle trgl(7,8);
    //Object Instantiation is not allowed with an abstract class.
    // Polygon poly(7,8);
    Polygon* ppoly = &rect;
    cout << ppoly->area() << endl;          //56 = 7x8
    ppoly = &trgl;
    cout << ppoly->area() << endl;          //28 = 7x8/2
    //ppoly = &poly;
    //cout << ppoly->area() << endl;        //0
    return 0;
  }
```

Polymorphism: Runtime

- **Benefits**
  - Decouples the names from the behaviors both at object and at function levels.
  - Adds more flexibilities to incorporate variations into a software API.
  - Makes a software API more maintainable.
  - Makes a software API more testable.
- **Costs**
  - Virtualization involves indirection during function calls and indirection makes the execution slow.