Polymorphism

Introduction

Polymorphism allows a single interface to represent different types. It enables code flexibility and reusability by allowing objects to behave differently based on their context via virtual functions.

Method Overriding

Inheritance allows a process called *function overriding* (runtime polymorphism) in which a derived class can redefine its base class methods. A derived class method overrides a base class method if their function signatures (function headers) are the same [or differ only by *covariant return types* (class pointer or reference)]. Afterward, derived class objects will use their overridden version of the method.

Virtual Methods

Although method overriding allows a derived class to provide a new implementation of an inherited method, if a derived class object is assigned to a base class object, pointer, or reference, the base class implementation is used since the derived class attributes or behaviors are lost. This process is called *object slicing*.

To prevent object slicing, enable dynamic dispatch by quantifying the base class method with the '`virtual`' keyword. This ensures that when a base class pointer or reference is used, the overridden method in the derived class is executed via the *virtual table* (vtable). The syntax for declaring a virtual method is:

`virtual` *method-definition*

An overridden method of a virtual method remains virtual. To explicitly indicate that a derived class method overrides a virtual method, use the '`override`' keyword in its definition:

method-header `override` {*body*}

And to prohibit further overrides, append the '`final`' keyword to its definition:

method-header `final` {*body*}

Abstract Classes & Interfaces

A *pure virtual function* (also known as an *abstract function*) is a virtual function that acts as a template for derived classes. It forces subclasses to provide their implementation. The syntax for declaring a pure virtual function is:

`virtual` *function-header* `= 0;`

A class containing at least one pure virtual function is called an *abstract class*. Abstract classes cannot be instantiated directly—only pointers and references to them can be created. A derived class is abstract if any inherited pure virtual functions are not defined.

Pure virtual methods can be overridden in derived classes regardless of their access specifier. An abstract class containing only pure virtual methods is called an *interface*. Inheriting multiple interfaces with at most one concrete or abstract class does not cause conflicts.

Virtual Destructor

If a base class contains at least one virtual function, its destructor should also be virtual as follows:

`virtual ~`*base-class* {*body*}

This ensures that when a derived object is deleted through a base class pointer or reference, its destructor is correctly invoked. Only the base class destructor runs without a virtual destructor, which leads to potential memory leaks if the derived class dynamically allocates resources.