Inheritance

Terms

Abstract classes
Base class
Child class
Derived class
Dynamic binding
Early binding
Final classes and methods
Inheritance
Late binding
Overriding a method
Parent class
Polymorphism
Protected members
Pure virtual methods
Redefining a method
Static binding

Summary

- *Inheritance* allows us to create a new class based on an existing class. The new class automatically inherits all the members of the base class (except constructors and destructor).
- A class that another class inherits is called a *base* or *parent class*.
- A class that inherits another class is called a *derived* or *child class*.
- *Protected members* of a class are accessible within a class and the derived classes, but not outside of these classes.
- Instances of a derived class can be implicitly converted to their base type because they contain everything the base class expects.

- A derived class can *redefine* a non-virtual method in the base class by providing its own version of that method.

- A derived class can *override* a virtual function in the base class by providing its own implementation of that function.

- *Polymorphism* refers to the situation where an object can take many different forms. This happens when we pass a child object to a function that expects an object of the base class.

- Redefining a function prevents polymorphic behavior because it forces the compiler to decide which function to call at compile time. This is known as *static or early binding*.

- Overriding a function enables polymorphism. It allows the compiler to determine which function to call at runtime. This is known as *dynamic* or *late binding*.

- A *pure virtual method* is a method that has to be overridden in a derived class. We can declare a method as a pure virtual method by coding "= 0" in its declaration.

- An *abstract class* is a class with at least one pure virtual method.

- Abstract classes cannot be instantiated. They exist just to provide common code to derived classes.

- *Final methods* cannot be overridden.

- *Final classes* cannot be inherited.

- *Multiple inheritance* allows a class to inherit multiple classes.

- While inheritance is a great technique for code reuse, deep inheritance hierarchies often result in complex, unmaintainable applications and should be avoided.


```
// Inheritance
class TextBox : public Widget {};

// Calling the constructor of the base class
TextBox::TextBox(bool enabled) : Widget(enabled) {}

// Implicit conversion to the base type.
// Object slicing happens here.
TextBox textBox;
Widget widget = textBox;


// No object slicing happens here because
// we're using a reference variable.
TextBox textBox;
Widget& widget = textBox;


// Method overriding
class Widget {
public:
    virtual void draw() const;
};

class TextBox : public Widget {
public:
    void draw() const override;
};
```



```cpp
// Abstract class
class Widget {
public:
    // Pure virtual method
    virtual void draw() const = 0;
};


class TextBox : public Widget {
public:
    // Final method
    void draw() const override final;
};


// Final class
class TextBox final : public Widget {
public:
};


// Multiple inheritance
class DateTime : public Date, public Time {};
```