# Polymorphism

# Contents

- Objectives
  - To be able to implement the third OO principle – Polymorphism.
- Contents
  - Virtual functions.
  - Vtables
  - Virtual Destructors
  - Pure virtual functions.
  - Be able to implement polymorphism.
  - Override.
  - Abstract base classes & Interface classes.

2

# Polymorphism

From Greek:

Poly – meaning many
Morph – meaning forms

3

Polymorphism in C++ is the mechanism whereby a single command can be used to implement different behaviours in different classes.

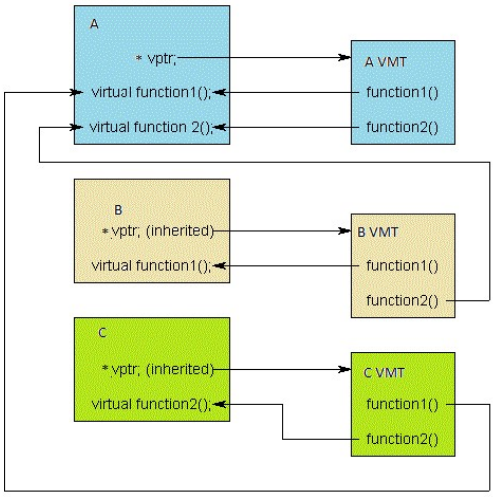
# Virtual Functions

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 class Insect
 5 {
 6 public:
 7     virtual void move() const
 8     {cout << "Generic move\n";}
 9 };
10
11 class Ant : public Insect
12 {
13 public:
14     void move() const
15     {cout << "crawl crawl\n";}
16 };
17
18 class Butterfly : public Insect
19 {
20 public:
21     void move() const
22     {cout << "flap flap\n";}
23 };
```

4

A function that is to be used as a polymorphic function in C++ must be declared as virtual in the base class. The keyword virtual should precede the function type in the function declaration. The function may also be declared as virtual in the derived class, although this is optional. The virtual function can be overridden in the derived class. However, it is not compulsory to do so. If a derived class does not override the virtual function then the function that will be called is that which is defined in the base class.

# Virtual Function Table




Every object of a class that contains a virtual function carries with it a Virtual Function Table (Vtable). RTTI is used to determine the objects type at run time. All of this is an overhead, but one which is usually worth suffering for the benefits that polymorphism can bring. The vtable determines which function will be called.

## Virtual Destructors

Base class destructors should be declared as virtual

6

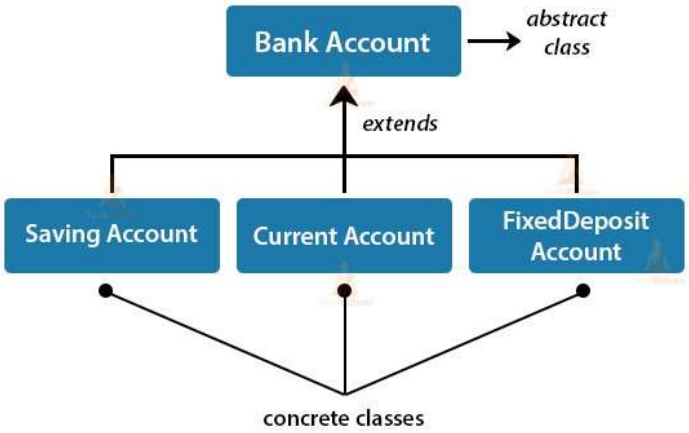
A delete command has two effects. To call the destructor for the object (if there is one defined) and to deallocate the memory used by the object.

When accessing derived classes via a pointer to the base class the compiler has only the base class definition to go by to work out what it needs to do. If the derived class creates objects on the heap dynamically this may not get picked up. Once again we can have memory leakage. Worse, if the base class allocates more memory than the derived class an attempt will be made to reclaim a larger area of memory than has been allocated. This can lead to all sorts of memory violation errors.

The solution is to declare the base destructor as virtual. This instructs the compiler to destruct and deallocate based on the lowest derived class. Destruction goes from the lowest derived class upwards.

Once a destructor has been described as virtual in a base class all destructors defined in derived classes will also be virtual.

# Pure Virtual Functions




7

Any class that has pure virtual functions is an abstract class. It can only be used to derive other specific classes. In other words, objects cannot be created from abstract classes. Any attempt to instantiate an abstract class will result in a compiler error. If the derived class does not have any pure virtual functions it is said to be concrete rather than abstract.

## Pure Virtual Functions

```
1 class Tree
2 {
3 public:
4     virtual void Grow() = 0;
5
6     virtual void ShedLeaves()
7     {
8         // shed leaves in autumn, shed and set seeds
9     }
10 };
```

8

Some classes are intended only to provide a common base class. It is not meaningful to create an abstract class object. In the Tree example there is no such thing as an Tree. There are, however, deciduous & evergreen trees etc. These are derived from the tree class and are real objects. Certain operations in an abstract class cannot meaningfully be defined. For example, the grow command for a tree. All trees have some operations in common & it is quite usual to see operations defined in abstract classes. They are used as placeholders to inform class users that derived classes will provide a member function that is specific to the derived class. Within the abstract class the operations are abstract.

A class that has only pure virtual functions – no state or data members is also known as an interface class. Although an advanced topic it is worthwhile noting that interfaces play an important part on OO technologies such as CORBA and COM.

# Exercise