C++ Non-Abstract Base Classes

- **Inheritance of interface only: Abstract Base Classes**
- **... of implementation only: private base classes (later)**
- **... of both interface, implementation: public base classes**

- **Public inheritance from a non-abstract base class**
  There are subtleties since constructors and other member methods are treated differently. Some care is required to ensure semantic relationships are maintained. Constructors and friendship are ***not*** inherited, member methods are.

- **Handle class [Stroustrup3] - a concrete class is the interface; a separate class represents the actual state**
  This is yet another means to allow the implementation details of a class to vary without changing the user-visible interface.
- **Policy class - an class whose only purpose is to add or refine the "policy" in another class**

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

Example: Trybos Banks

- **Trybos Generic Bank -** is a contiguous array of integers. The bank is identified by a (name, number) pair called a key. The format of the contained user data depends on the name of the bank. A type description is included which describes the data types of user data to allow for machine byte order adjustments. Generic Bank is the concrete base class which implements the Trybos file and record storage capability. It also provides a consistent set of data access services for individual bank classes to re-use. Trybos Generic Bank is a ***handle*** class in order to hide the differences in accessing banks allocated locally and banks stored in a Trybos record.
- **BNK Named Bank -** is a specialization of Trybos Generic Bank. Named bank requires each child class to have a ***static*** bank name defined which is checked against the name of banks assigned to it. Named Bank class itself is an abstract ***policy*** class. Users cannot create instances of it, only derive from it.
- **CMUO Bank -** is a named bank storing data for a Central Muon Object. It defines its class name and adds an interface to the user data stored in it.

Trybos Banks Interface...

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

```
class TRY_Generic_Bank
{ public:
  TRY_Generic_Bank(bank_key, bank_type) ;  \\ Basic method
  virtual void assign(bank_key, bank_type) ;      \\ Basic method
   int size(void) const ;                         \\ Used by all children as is
}  ;

class BNK_Named_Bank: public TRY_Generic_Bank
{ public:
  BNK_Named_Bank(bank_key, bank_type) ;  \\ Refined method, test name
  virtual void assign(bank_key, bank_type) ;    \\ Refined method, test name
  static class_bank_name(void) const = 0 ;    \\ Require this method of children
} ;

class CMUO_Bank: public BNK_Named_Bank
{ public:
  CMUO_Bank(bank_number) ;          \\ Refined further: known name and type
  virtual void assign(bank_number) ;   \\ Refined further: known name and type
  static class_bank_name(void) const ;  \\ Implement this method here
} ;
```


And Their Implementation...

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

```
class TRY_Generic_Bank
{ public:
  TRY_Generic_Bank(bank_key, bank_type) ;  \\ Allocate memory, fill values
  virtual void assign(bank_key, bank_type) ;      \\ Allocate memory, fill values
  int  size(void) const ;                     \\ Implemented in compiled source file
}  ;

class BNK_Named_Bank: public TRY_Generic_Bank
{ public:
  BNK_Named_Bank(key, type): TRY_Generic_Bank(key, type) \\base class init
       { if (class_bank_name() != key.bank_name())  handle_error() ; }
  virtual void assign(key, type)
       { if (class_bank_name() != key.bank_name())  handle_error() ;
         TRY_Generic_Bank::assign(key, type) ;
       }
\\ int  size(void) const ;    ALWAYS INHERITED AS IS, so not declared virtual
  static virtual class_bank_name(void) const = 0 ;    \\ Required of children
} ;
```

More Implementation...

```
class CMUO_Bank: public BNK_Named_Bank
{ public:
  CMUO_Bank(bank_number): BNK_Named_Bank(
                              TRY_Bank_Key(class_bank_name(), bank_number),
                              class_bank_type() )  \\ this method not shown here
  virtual void assign(bank_number)
    { BNK_Named_Bank::assign(
                              TRY_Bank_Key(class_bank_name(), bank_number),
                              class_bank_type() ) ;
    }
\\ int  size(void) const ;    ALWAYS INHERITED AS IS, so not declared virtual
static virtual class_bank_name(void) const  { return(_class_bank_name) ; }
private:
static TRY_Bank_Name   _class_bank_name ;
} ;
```

<u>And to the source file</u> CMUO_Bank.cc, <u>we add the initializer statement</u>

```
TRY_Bank_Name  CMUO_Bank::_class_bank_name("CMUO") ;
```


Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

And Their Use.

```
TRY_Bank_Key       test_key("TEST", 1) ;            \\ Ignore details for now
TRY_Bank_Type      test_type("(23I4,46R4)") ;       \\ Ignore details for now
TRY_Generic_Bank test_bank(test_key, test_type) ; \\same size as CMUO bank
CMUO_Bank          muon_bank(2) ;                   \\ bank number == 2

if (test_bank.size() == muon_bank.size())     \\ method inherited from base class
  {  cout << "They are the same size" << endl ; }

CMUO_Bank muon_test(test_bank) ;          \\ But their bank names differ
if (muon_test.is_invalid())
  { cout << "muon_2 construction: input bank has the wrong name" << endl ;
  }

muon_test.assign(test_bank) ;                   \\ Their bank names still differ
if (muon_test.is_invalid())
  { cout << "muon_2 assignment: input bank has the wrong name" << endl ;
  }
```

Private Inheritance Example

- **Trybos IO Buffer -** is a contiguous array of integers. Unlike a Trybos integer array (TRY_Vector<int>), it does not have distinct logical and storage size concepts; it only has a storage size. The object using it maintains the logical size concept. Thus, the interface of an IO Buffer will be very different from that of an integer array (no interface inheritance). The well-tested integer array code can still be re-used if IO Buffer privately inherits from integer array.

```
#include "Trybos/TRY_Vector.hh"
class TRY_IO_Buffer: private TRY_Vector<int>
{ public:  \\ Much code missing...
  int& operator[](const int index) \\Just forward the call to the private base class
     { return( TRY_Vector<int4>::operator[](index) ) ; }
  const int& operator[](const int index) const
     { return( TRY_Vector<int4>::operator[](index) ) ; }
} ;
```


Summary

- **Interface re-use through public inheritance from an abstract base class:** Clean separation of interface and implementation into different classes, with commonality percolated up to abstract base class. Implementation classes are interchangeable at run-time.
- **Interface and implementation re-use through public inheritance:** Each child class refines the interface of the base class, while re-using the implementation of the base class. This allows, as with the Trybos bank classes, a complicated interaction between the base class with other classes (Trybos record classes) to be re-used by specialized children.
- **Implementation re-use through private inheritance:** A valuable implementation can be re-used by a very different class. Experts usually advice, however, to consider using ***containment*** instead of private inheritance. Applied to our example, TRY_IO_Buffer would contain an integer array data member and ***forward*** its operations to the data member as before.