Layering a Component A Detailed Example Using the Queue

Part 3 – Creating Different Implementations

```
// Filename: Queue2.hpp
#pragma once
#include "List\List1.hpp"

template <class T>
class Queue2 : public StdOps<Queue2<T>>,
               public QueueKernel<T>
{
public: // Standard Operations
   Queue2();
   ~Queue2();
   void clear (void);
   void transferFrom (Queue2& source);
   Queue2& operator = (Queue2& rhs);
// Queue2 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replacefront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   ...
};
```

Create a Different Component Implementation

Different implementations – each component can be implemented multiple different ways

How is this done?

- Each implementation must have the same abstract public interface and external contracts
- But will have a different private concrete representation and implementation
- The compiler enforces conformance to the exact same syntactic interface because all QueueX components must inherit from StdOps and QueueKernel abstract classes

Example of a Different Component Implementation

Queue3 layered on Sequence1

```
// Filename: Queue3.hpp
#pragma once
#include "Sequence\Sequence1.hpp"

template <class T>
class Queue3 : public StdOps<Queue3<T>>,
               public QueueKernel<T>

{

public: // Standard Operations

   Queue3();

   ~Queue3();

   void clear (void);

   void transferFrom (Queue3& source);

   Queue3& operator = (Queue3& rhs);

// Queue3 Specific Operations

   void enqueue (T& x);

   void dequeue (T& x);

   void replacefront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   typedef Sequence1<T> SequenceOfT;
   SequenceOfT s;

};
// Member functions manipulate
// SequenceOfT s
```

The **highlighted** parts (to the left) show what has changed from Queue2's implementation – where Queue2 was layered on List1

In the top part of the file:

1. Give the file a different name, e.g., Queue3.hpp
2. #include Sequence instead of List

In the public part:

3. Use a different template class name, e.g., Queue3

In the private part:

4. Create instance of Sequence using template parameter T
5. Declare a data member from instance of Sequence

In the member function part:

6. Implement the member functions so that they store the data in the Sequence data member