Queue

An abstraction for a
first in first out data structure

The Queue Component

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Shown here using C++'s *template class* construct

The Queue Component

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

C++'s *template* construct means that Queue1 is parameterized on the data type to be stored in the queue

Here the parameter name is *T*

Prior to use, the client programmer must instantiate the template by supplying the name of the data type to be stored in the queue

The Queue Component

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Queue has 10 member functions:

- The 5 *Standard Operations*
- And 5 *Queue Specific Operations*

Standard Operations

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

← *Standard* means that each C++ component always exports these 5 operations

1. constructor
2. destructor
3. clear
4. transferFrom
5. operator =

Specific Operations

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Each C++ component also exports operations that give the component its unique behavior – these are the *component specific operations*

There are 5 operations that are specific to Queue which give it first in, first out behavior (or FIFO behavior):

1. enqueue
2. dequeue
3. replaceFront
4. front
5. length

Side by Side Comparison of Two Components – Queue & Map

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

```
template <class K, class V>
class Map1
{
public: // Standard Operations
   Map1();
   ~Map1();
   void clear (void);
   void transferFrom (Map1& source);
   Map1& operator = (Map1& rhs);
// Map1 Specific Operations
   void add (K& key, V& value);
   void remove (K& key, K& k, V& v);
   V& value (K& key);
   void removeAny (K& k, V& v);
   Boolean hasKey (K& key);
   Integer size (void);
private: // representation
   // ...
};
```

Queue's Abstract Model

```
template <class T>
class Queue1
  //! is modeled by string of T
  //!   exemplar self
{
public: // Standard Operations
  Queue1();
  ~Queue1();
  void clear (void);
  void transferFrom (Queue1& source);
  Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
  void enqueue (T& x);
  void dequeue (T& x);
  void replaceFront (T& x);
  T& front (void);
  Integer length (void);
private: // representation
  // ...
};
```

Queue is modeled by string of T

T is the template parameter

This part of the spec indicates that we reason about queue variables as math strings of type T

```
template <class T>
class Queue1
   //! is modeled by string of T
   //!   exemplar self
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Reasoning Using The Abstract Model

```
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1;
```

For example – on these 2 lines of client code:

The *typedef* is used to instantiate Queue1 where *T*'s actual parameter is *Integer*

On the second line the variable q1 declared

We reason about q1's value by thinking of it as a string of Integers

Below is an example of q1's value after enqueueing the integers: 33, 71, 10

q1 = <33,71,10>

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

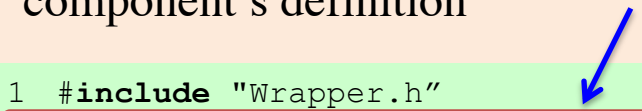
A Client of Queue1

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Queue1 Template Instantiation

In order to use a component, the client must #include the file containing the the component's definition



```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Queue1 Template Instantiation

The client uses C++'s *typedef* construct to instantiate Queue1 with type Integer

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

Template Instantiation

The C++ compiler substitutes Integer for the template parameter *T* everywhere in Queue1

```
template <class T>
class Queue1
   //! is modeled by string of T
   //!   exemplar self
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

Type Name Substitution

Queue1 after the substitution of *Integer* for *T*

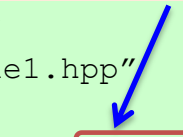
```
class Queue1
   //! is modeled by string of Integer
   //!   exemplar self
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue(Integer& x);
   void dequeue(Integer& x);
   void replaceFront (Integer& x)
   Integer& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Named Instantiation

typedef also allows the programmer to provide a name for the instantiation – in this example *IntegerQueue* is the given name



```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Variable Declaration

IntegerQueue is then used in this client program to declare two queue variables: q1 and q2

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Variable Usage

q1 and q2 are *IntegerQueues*, so only Integer values can be stored in q1 and q2

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```

The *exemplar*

```
template <class T>
class Queue1
   //! is modeled by string of T
   //!   exemplar self
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

exemplar self also appears in Queue's spec

self is used to refer to the queue variables that appear before the dot in executable statements in a client program

controlling object

- is how we will refer to the object in front of the dot
- For example:
  `q1.enqueue(x1);`
  q1 is the controlling object

self Example

In the example below, we focus only on the calls to *enqueue*. In each of these calls, *self* refers to the queue variable before the dot

```
template <class T>
class Queue1
   //! is modeled by string of T
   //!   exemplar self
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

controlling objects

```
1  #include "Wrapper.h"
2  #include "Queue\Queue1.hpp"
3
4  typedef Queue1<Integer> IntegerQueue;
5
6  int main(int argc, char* argv[])
7  {
8     IntegerQueue q1, q2;
9     Integer x1, x2;
10
11    x2 = 33;
12    q1.enqueue(x2);
13    x1 = 44;
14    q1.enqueue(x1);
15
16    q2.transferFrom(q1);
17    x1 = 55;
18    q2.enqueue(x1);
19
20    cout << q1.length();
21
22    q2.dequeue(x2);
23    q1 = q2;
24    q2.clear();
25 }
```