```
    Queue & operator=(const Queue & q) { return *this;}
//...
} ;
```

This has two effects. First, it overrides the default method definitions that otherwise would be generated automatically. Second, because these methods are private, they can't be used by the world at large. That is, if `nip` and `tuck` are `Queue` objects, the compiler won't allow the following:

```
Queue snick(nip);      // not allowed
tuck = nip;            // not allowed
```

Therefore, instead of being faced with mysterious runtime malfunctions in the future, you'll get an easier-to-trace compiler error stating that these methods aren't accessible. Also, this trick is useful when you define a class whose objects should not be copied.

Are there any other effects to note? Yes. Recall that the copy constructor is invoked when objects are passed (or returned) by value. However, this is no problem if you follow the preferred practice of passing objects as references. Also, the copy constructor is used to create other temporary

objects. But the `Queue` definition lacks operations that lead to temporary objects, such as overloading the addition operator.

The Customer Class

Next, you must design a customer class. In general, a teller machine customer has many properties, such as a name, account numbers, and account balances. However, the only properties you need for the simulation are when a customer joins the queue and the time required for the customer's transaction. When the simulation produces a new customer, the program will create a new customer object, storing in it the customer's time of arrival and a randomly generated value for the transaction time. When the customer reaches the front of the queue, the program will note the time and subtract the queue-joining time to get the customer's waiting time. Here's how you can define and implement the `Customer` class:

```
class Customer
{
private:
    long arrive;          // arrival time for customer
```

```
    int processtime;    // processing time for customer
public:
    Customer() { arrive = processtime = 0; }
    void set(long when);
    long when() const { return arrive; }
    int ptime() const { return processtime; }
} ;
void Customer::set(long when)
{
    processtime = rand() % 3 + 1;
    arrive = when;
}
```

The default constructor creates a null customer. The `set()` member function sets the arrival time to its argument and randomly picks a value from 1 through 3 for the processing time.

Listing 12.8 gathers together the `Queue` and `Customer` class declarations, and Listing 12.9 provides the methods.