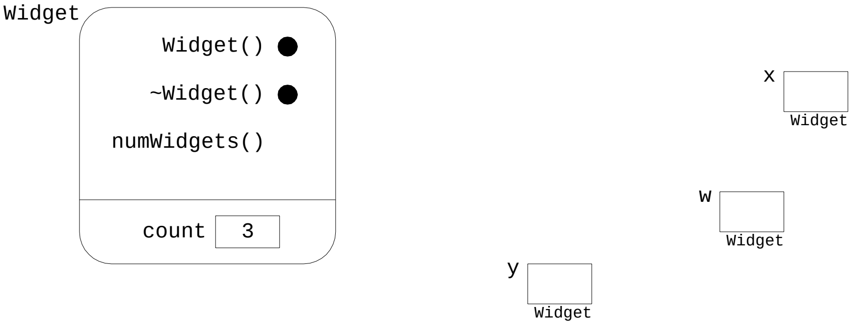
```
   ~Widget() { --count; }
   static int num() { return count; }
  private:
   static int count;
 };

 int Widget::count = 0;

 int main()
 { cout << "Now there are " << Widget::num() << " widgets.\n";
  Widget w, x;
  cout << "Now there are " << Widget::num() << " widgets.\n";
  { Widget w, x, y, z;
   cout << "Now there are " << Widget::num() << " widgets.\n";
  }
  cout << "Now there are " << Widget::num() << " widgets.\n";
  Widget y;
  cout << "Now there are " << Widget::num() << " widgets.\n";
}
```

Declaring the `num()` function to be `static` renders it independent of the class instances. So now it is invoked simply as a member of the `Widget` class using the scope resolution operator “`::`”. This allows the function to be called before any objects have been instantiated.

The previous figure showing relationships among the class and its instances should now looks like this:



The difference is that now the member function `num()` has no “`this`” pointer. As a `static` member function, it is associated with the class itself, not with its instances.

Static member functions can access only `static` data from their own class.

Review Questions

10.1 Explain the difference between a `public` member and a `private` member of a class.
10.2 Explain the difference between the interface and the implementation of a class.
10.3 Explain the difference between a class member function and an application function.
10.4 Explain the difference between a constructor and a destructor.
10.5 Explain the difference between the default constructor and other constructors.
10.6 Explain the difference between the copy constructor and the assignment operator.
10.7 Explain the difference between an access function and a utility function.
10.8 Explain the difference between a `class` and a `struct` in C++.