Introduction

It is desirable, and at times indispensable, for all instances of a class to share the literal value of a particular class member. In C++ this notion is referred to as static member. Manipulation of static members requires defining static methods. In C++ a static member is manipulated without reference to any instance of the class owning the member.

The abuses of C++ linguistic presentation of the notion (static member) were not in the designer's agenda. Consider the popular usage of declaring all members and methods of a class as static. This effectively reduces the class to a namespace. This is particularly popular among Java programmers for implementing enumerations.

Z++ is an evolutionary superset of C++ for developing platform-free applications. It extends C++, but it also corrects known issues. In this paper we illustrate how Z++ prevents abuses of static members. This notion is called common in Z++. Thus, a common member in Z++ is conceptually identical to a static member in C++.

Declaring a common member

The following is the syntax for declaring a common member.

```
common Type Member : prototype_1, …, prototype_n ;
```

The term common is a keyword, Type is a type-name and Member stands for an identifier for the data member. So far, this is the same as declaring a C++ static member. In fact, the sequence of prototypes is optional, and the colon is only required when the sequence is not empty.

A common member cannot be public. The prototypes after the colon are prototypes of methods of the class in which the common member is being declared. These are called the authorized methods (associated with the common member).

A common member is initialized the same way as in C++. Thus, before an instance of a class is created, all its common members will have been created and initialized.

Authorized Methods

The only methods that can modify a common member are those in the list of its authorized methods. There are no common methods in Z++. Therefore, common members can only be manipulated via instances of the class. That is, first an instance of the class must be created. Then, authorized methods can manipulate their associated common members.

Constructors and destructors of a class cannot be associated with its common members. However, all methods, as well as constructors and destructors, can access common members.

Conclusion

The Z++ linguistic presentation of common members provides all the benefits of the notion of C++ static members. Furthermore, it blocks the uses that distort the abstraction.

The abstraction is that, any change to a common member of a class is reflected to all instances of the class. However, like any other member, a common member should only be reachable through an instance of the class that owns the member.

The list of authorized methods associated with a common member specifies the only methods that can modify the common member. This ensures that a common member is not changed accidentally.