- You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.
- You want to use an object in an environment that expects an interface that is different from the object's interface.
- Interface translation among multiple sources must occur.

Bridge Pattern

The Bridge pattern divides a complex component into two separate but related inheritance hierarchies: the functional abstraction and the internal implementation. This makes it easier to change either aspect of the component so that the two can vary independently.

The Bridge pattern is useful when there is a hierarchy of abstractions and a corresponding hierarchy of implementations. Rather than combining the abstractions and implementations into many distinct classes, the Bridge pattern implements the abstractions and implementations as independent classes that can be combined dynamically. Figure 4–7 illustrates the Bridge pattern.

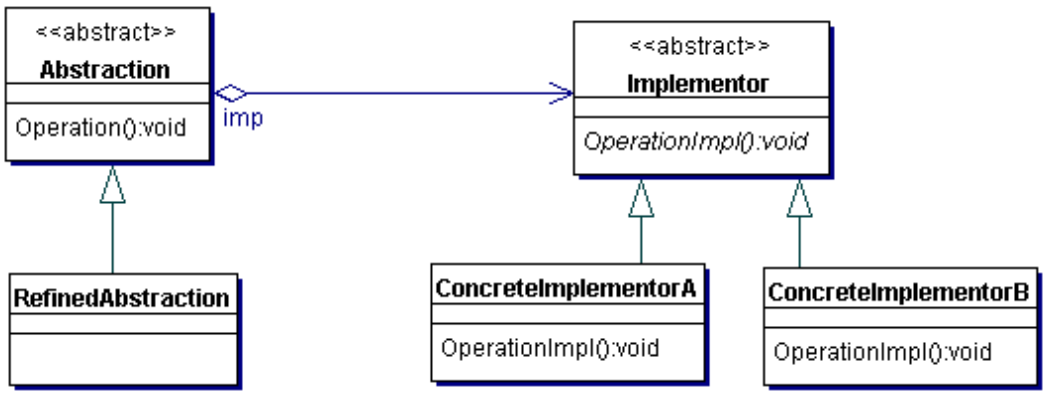

Figure 4–7 *The Bridge Pattern*

BENEFITS

The following lists the benefits of using the Bridge pattern:

- Enables you to separate the interface from the implementation
- Improves extensibility
- Hides implementation details from clients