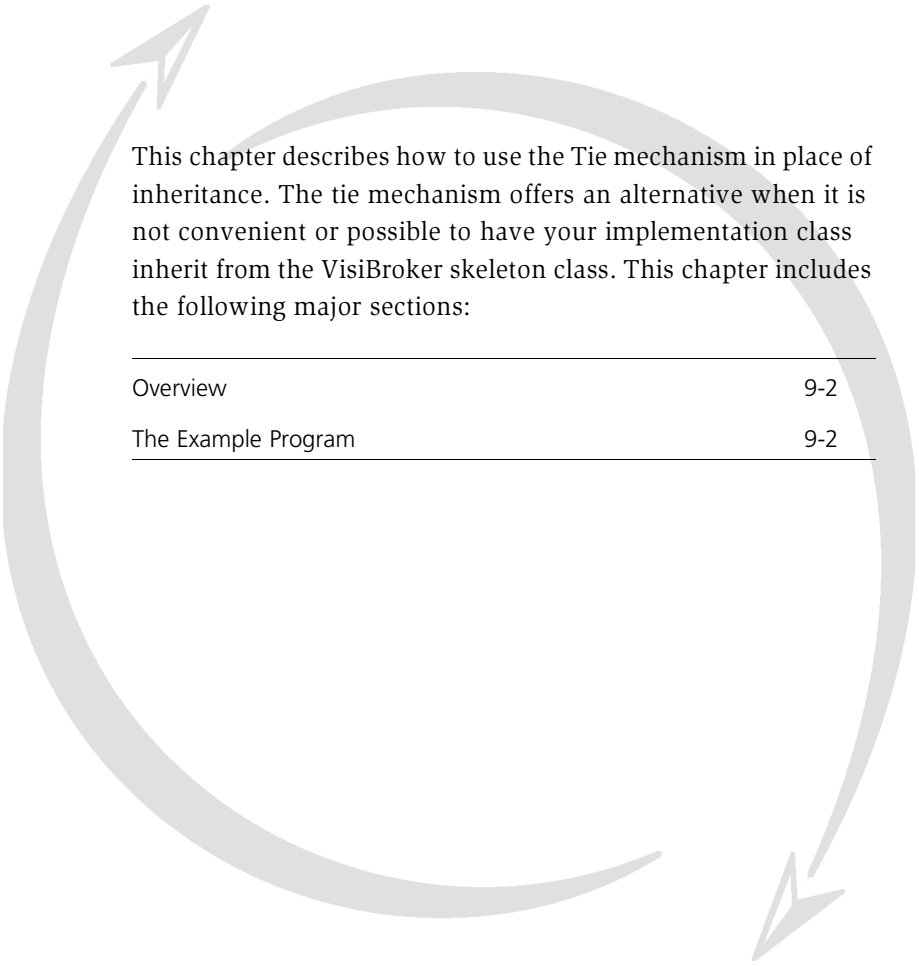
CHAPTER 9

THE TIE MECHANISM



This chapter describes how to use the Tie mechanism in place of inheritance. The tie mechanism offers an alternative when it is not convenient or possible to have your implementation class inherit from the VisiBroker skeleton class. This chapter includes the following major sections:

Overview	9-2
The Example Program	9-2

9 THE TIE MECHANISM

OVERVIEW

The tie mechanism offers an alternative when it is not convenient or possible to have your implementation class inherit from the VisiBroker skeleton class. Object implementation classes must inherit from the VisiBroker skeleton class. Java does not allow multiple class inheritance and you may want your implementation class to inherit from a different class.

Inheritance is easier to use than the tie mechanism because implementation objects look and behave just like object references. If an implementation object happens to be in the same process as a client using it, operation invocations involve less overhead because no transport, indirection, or delegation of any kind is required.

The tie mechanism provides a *delegator implementation* class that inherits from `org.omg.CORBA.Object`. The delegator implementation does not provide any semantics of its own. It simply delegates every call to the real implementation class, which can be implemented separately and can inherit from whichever class it wishes.

The `Operations` class defines all of the methods that must be implemented by the object implementation. This class acts as the delegate object for the associated `_tie` class when the tie mechanism is used.

THE EXAMPLE PROGRAM

A version of the `Bank` example using the tie mechanism can be found in the VisiBroker for Java distribution in the **bank_using_tie** directory within the **java_examples** directory. Only the **Server.java** file is different from the simple bank example.

The `_tie_Account` and `_tie_AccountManager` files are automatically generated by the `idl2java` compiler. These are delegator implementations for the interfaces `Account` and `AccountManager`. Each instance of the `_tie` class must be initialized with an instance of a real implementation class to which it delegates every operation. In this example, the class `Account` does not inherit from the `_AccountImplBase` class, nor does the class `AccountManager` inherit from the `_AccountManagerImplBase` class. Both of these classes are now free to inherit from another class.

Building the Tie Example

The instructions described in "Building the Example Programs" on page 2-13 are also valid for building the tie example.

Changes to the Server Class

Code Sample 9-1 shows the modifications to the `Server` class. Note the extra step of creating an instance of `_tie_AccountManager`.

- Create an `AccountManager`.
- Create a `_tie_AccountManager` and initialize it with the `AccountManager`.
- Export the `_tie_AccountManager` object.
- Wait for requests.

CODE

```
...

public class Server {
  public static void main(String[] args) {
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    org.omg.CORBA.BOA boa = orb.BOA_init();
    // create the account manager
    AccountManagerImpl new_manager =
      new AccountManagerImpl(boa);
    Bank.AccountManager manager =
      new Bank._tie_AccountManager(new_manager,"BankManager");
    // export the object reference
    boa.obj_is_ready(manager);
    System.out.println(manager + " is ready.");
    // wait for requests
    boa.impl_is_ready();
    }
}
```

*Code Sample 9-1 The `Server` class portion of the **Server.java** file from the **bank_using_tie** directory.*

Changes to the AccountManager

The changes made to the `AccountManager` class include:

- `AccountManager` no longer extends `Bank._AccountManagerImplBase.`
- When a new `Account` is to be created, a `_tie_Account` is also created and initialized.
- The `_tie_Account` is exported.

Changes to the Account Class

The only changes that are made to the `Account` class are that it no longer extends `Bank._AccountImplBase`.

CODE

```
// Server.java
import java.util.*;
class AccountImpl implements Bank.AccountOperations {
  AccountImpl(float balance) {
    _balance = balance;
  }
  public float balance() {
    return _balance;
  }
  private float _balance;
}
```

Code Sample 9-2 *The* `AccountImpl` *class portion of the* ***Server.java*** *file in the* ***bank_using_tie*** *directory.*

CODE

```
class AccountManagerImpl implements Bank.AccountManagerOperations {
    private org.omg.CORBA.BOA _boa;

    public AccountManagerImpl(org.omg.CORBA.BOA boa) {
      _boa = boa;
    }

    public Bank.Account open(String name) {
      // lookup the account in the account dictionary
      Bank.Account account = (Bank.Account) _accounts.get(name);
      if(account == null) {
        // create a new account with 0-1000 dollars!
        float balance = Math.abs(_random.nextInt()) % 100000 / 100f;
        AccountImpl new_account = new AccountImpl(balance);
        // create the tie object for the account
        account = new Bank._tie_Account(new_account);
        System.out.println("Created " + name + "'s account: " + account);
        // export the object reference
        _boa.obj_is_ready(account);
        // save the account in the account dictionary
        _accounts.put(name, account);
      }
      return account;
```

CONTINUES ON NEXT PAGE

CODE CONTINUED FROM PREVIOUS PAGE

```
    }
    private Dictionary _accounts = new Hashtable();
    private Random _random = new Random();
}
```

Code Sample 9-3 *The* `AccountManagerImpl` *class portion of the* ***Server.java*** *file in the* ***bank_using_tie*** *directory.*