## Contents

### JADE Interfaces

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>The Interface Browser</td>
<td>4</td>
</tr>
<tr>
<td>Defining an Interface</td>
<td>5</td>
</tr>
<tr>
<td>Implementing an Interface</td>
<td>7</td>
</tr>
<tr>
<td>Exercise 1 – Adding the IWithdrawable Interface</td>
<td>10</td>
</tr>
<tr>
<td>Exercise 2 – Implementing the IWithdrawable Interface</td>
<td>11</td>
</tr>
<tr>
<td>Exercise 3 – Testing the IWithdrawable Interface</td>
<td>14</td>
</tr>
</tbody>
</table>
Introduction

An interface is a mechanism that provides a set of methods guaranteed to be available on any implementing class. When a class implements an interface, it agrees to implement all methods defined by the interface. With this contract in place, the implementing classes can participate in useful type-safe callback mechanisms.

Exposing a class via its interface effectively hides the implementation details of the class, as the user is forced to communicate only through the public interface methods.

Classes that implement an interface can be grouped by that interface type; for example, a collection can have a membership of a specified interface.

Using interfaces, non-related classes can be grouped to capture their similarities, without the need to artificially force a class relationship. Think of this as allowing a class to perform multiple roles outside of the role dictated by its class hierarchy.

**Tip** Interfaces are a useful way to group classes by behavior.

The Interface Browser

JADE allows for the defining of interfaces through the **Interface Browser**, in which you can specify the required methods and constants.

The following actions open the Interface Browser.

- Clicking on the **Browse Interfaces** icon in the browser toolbar.
- Selecting the **Interfaces** command from the Browse menu.
- Pressing Ctrl+N.
An Interface Browser is then opened for the selected schema.

The pane at the upper left contains a list of all interfaces defined for the selected schema, the middle pane displays any constants defined for the selected interface, and the pane at the upper right displays the required methods of the selected interface.

Although you may notice that the Interface Browser is similar to the Class Browser, note the following differences.

- Interfaces can define constants and methods only.
- Interface methods contain only the method signature; they do not have any implementation.
- Interfaces are always displayed in alphabetical order, with a flat (rather than nested) structure.
- While classes can extend a single superclass only, interfaces can extend any number of other interfaces.

**Defining an Interface**

The following actions create a new interface from the Interface Browser.

- Select the **Add** command from the Interfaces menu.
- Right-click in the pane at the upper left and then select the **Add** command from the pop-up (context) menu.
The Define Interface dialog is then displayed.

Enter a unique name for the interface, and whether to extend from one of the existing interfaces. To create the interface, click **Next** to create the interface while keeping the Define Interface dialog open (so that you can create multiple interfaces) or click **OK** to create the interface and close the Define Interface dialog.

**Note** Most of the normal options for creating a method are disabled for interface methods, because an interface contains the method signatures only. It is up to implementing classes to provide the method sources.
To add parameters or return types to an interface method, select the method in the Interface Browser and then modify the method signature.

Note  Attempting to provide a method body in an interface method will result in a compiler error.

Implementing an Interface

For a class to be able to implement an interface, it must:

- Explicitly indicate that it will provide the specific interface.
- Fulfil all methods of the interface.

To indicate the interfaces a class will implement, select the class in the Class Browser and then open the Interface Implementation Mapper dialog by doing either of the following.

- Selecting the Interface Mapping command from the Classes menu.
- Right-clicking the class in the Class Browser and then selecting the Interface Mapping command from the popup menu.
To choose an interface to add to the class, select it in the **Available Interfaces** drop-down list box.

The details of the interface will be displayed, including the signature of the interface methods and the types and values of the interface constants.
For each of the interface methods, the class must provide an implementation. By default, these will be automatically generated method stubs called `stub_method-name`. However, you can click on any of these to select an existing method from the class.

**Note**  The class method used to implement an interface method does not need to have the same name as the interface method; only the same parameters and return type. A single class method can fulfil the implementation of multiple interface methods if they have compatible method signatures.
To confirm the implementation, click Implement. The interface will then be added to the class, and by clicking on the Interface tab in the Methods List (the upper right pane) in the Class Browser, you can see the interface methods.

![Image showing the Class Browser and interface implementation](image)

**Note** Any interface method with which no class method was mapped has a stub implementation.

**Exercise 1 – Adding the IWithdrawable Interface**

In this exercise, you will add an interface to BankingModelSchema that provides an alternative to the polymorphism approach used for Accounts in Module 9 of the JADE Developer's course.

1. Select BankingModelSchema in the Schema Browser and then open the Interface Browser.
2. Add a new interface called IWithdrawable to the schema.
   
   **Note** Interfaces typically have a prefix of I or a suffix of IF.
3. Add a method called withdraw to the IWithdrawable interface.
4. Code the `withdraw` method signature, as follows.

![Interface Browser](image)

### Exercise 2 – Implementing the IWithdrawable Interface

In this exercise, you will implement a variety of methods that all fulfill the `withdraw` interface method in different ways.

1. Add a method called `attemptWithdrawal` to the `ChequeAccount` class and code it as follows.

```pascal
attemptWithdrawal(amount : Decimal) updating:
begin
  write "Cheque Balance: " & self.balance.String;
  write "Overdraft limit: " & self.overdraftLimit.String;
  write "Requested Funds: " & amount.String;
  if amount <= self.balance + self.overdraftLimit then
    beginTransaction;
    self.balance := self.balance - amount;
    commitTransaction;
    write amount.String & " withdrawn successfully."
  else
    write "Insufficient funds: " & (self.balance + self.overdraftLimit).String & " Available"
  end;
end;
```
2. Select the **ChequeAccount** class in the Class Browser and then open the Interface Implementation Mapper dialog by selecting the **Interface Mapping** command in the Classes menu.

3. Select **IWithdrawable** in the **Available Interfaces** drop-down list box and then enter **attemptWithdrawal** for the **withdraw** method in the **Jade Mapping** column. Click **Implement**, to confirm your action and close the window.

<table>
<thead>
<tr>
<th>Interface Names</th>
<th>Jade Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>withdraw</td>
<td>attemptWithdrawal</td>
</tr>
</tbody>
</table>

**Note** While interface methods themselves may not have the **updating** method option, **updating** methods can fulfil interface methods.
4. Add a method called `attemptWithdrawal`, this time to the `SavingsAccount` class, and code it as follows.

```pascal
attemptWithdrawal(amount : Decimal) updating;
begin
  write "Requested Funds: " & amount.String;
  if amount <= self.balance then
    beginTransaction;
    self.balance := self.balance - amount;
    commitTransaction;
    write amount.String & " withdrawn successfully.";
    write self.balance.String & " remaining";
  else
    write "Insufficient funds: " & (self.balance).String & " remaining";
  endif;
end;
```

5. Select the `SavingsAccount` class in the Class Browser and then open the Interface Implementation Mapper dialog by selecting the `Interface Mapping` command in the Classes menu.

6. Repeat Step 3 of this instruction.

7. Create a new class called `MoneyPit` in `BankingModelSchema`, add a method called `giveMoney`, and code it as follows.

```pascal
giveMoney(amount : Decimal);
begin
  write "Okay, have $" & amount.String;
end;
```

**Note** The `giveMoney` method is not an updating method. An interface method can have an updating or non-updating method fulfil it.
8. Select the **MoneyPit** class in the Class Browser and open the Interface Implementation Mapper dialog by selecting the **Interface Mapping** command in the Classes menu.

   ![Interface Implementation Mapper](image)

9. Select **IWithdrawable** in the **Available Interfaces** drop-down list box and then enter **giveMoney** for the **withdraw** method in the **Jade Mapping** column. Click **Implement**, to confirm your action and close the window.

---

**Exercise 3 – Testing the IWithdrawable Interface**

In this exercise, you will write a **JadeScript** method that puts the three classes that implement **IWithdrawable** interface into a collection and then call the **withdraw** method of each one.

1. Add a new **ObjectArray** subclass called **Withdrawables**, with member type **IWithdrawable**.
2. Add a new `JadeScript` method called `testInterface`, and code it as follows.

```jade
vars
dict  : Withdrawables;
acc   : IWithdrawable;
pit   : MoneyPit;

begin
create pit transient;
create dict transient;

dict.add(pit);
dict.add(ChequeAccount.firstInstance().IWithdrawable);
dict.add(SavingsAccount.firstInstance().IWithdrawable);
foreach acc in dict do
    acc.withdraw(45.50);
    write "-------------";
endforeach;
epilog
    delete pit;
    delete dict;
end:
```

3. Run the method. Output similar to the following should be displayed.

```
Okay, have $45.5

Cheque Balance: 0.00
Overdraft Limit: 500.00
Requested Funds: 45.5
45.5 withdrawn successfully.
-45.50 remaining

Savings Balance: 100.00
Requested Funds: 45.5
45.5 withdrawn successfully.
54.50 remaining
```