

UML for .NET Developers

Class Diagrams



Accelerated learning for C# developers

Get the UML for .NET Premium Package at
<http://www.parlezuml.com/tutorials/umlfordotnet.htm>

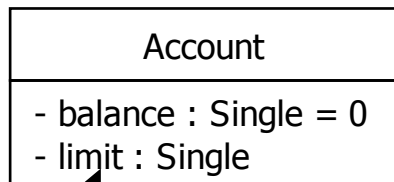
Classes

Account

```
class Account  
    {  
    }  
}
```

```
Class Account  
  
End Class
```

Attributes



```
class Account
{
    private float balance = 0;
    private float limit;
}
```

[visibility] [/] attribute_name[multiplicity] [: type [= default_value]]

```
Class Account

Private balance As Single = 0
Private limit As Single

End Class
```

Operations

Account
- balance : Single = 0 - limit : Single
+ deposit(amount : Single) ← + withdraw(amount : Single)

[visibility] op_name([[in|out] parameter : type[, more params]])[: return_type]

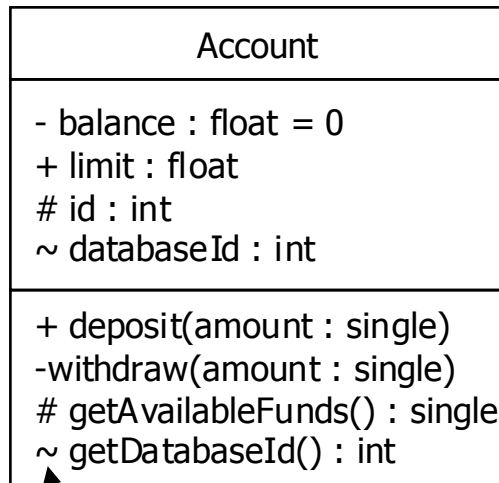
```
class Account
{
    private float balance = 0;
    private float limit;
    public void deposit(float amount)
    {
        balance = balance + amount;
    }

    public void withdraw(float amount)
    {
        balance = balance - amount;
    }
}
```

```
Class Account
    Private balance As Single = 0
    Private limit As Single = 0

    Public Sub deposit(ByVal amount As Single)
        balance = balance + amount
    End Sub

    Public Sub withdraw(ByVal amount As Single)
        balance = balance - amount
    End Sub
End Class
```



+ = public
 - = private
 # = protected
 ~ = package

Visibility – C#

class Account

```

{

    private float balance = 0;
    public float limit;
    protected int id;
    internal int databaseId;

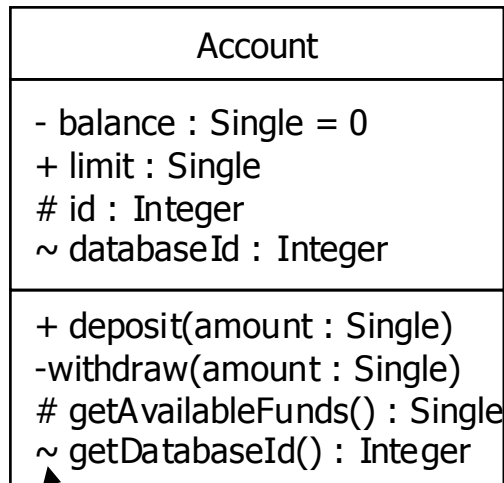
    public void deposit(float amount)
    {
        balance = balance + amount;
    }

    private void withdraw(float amount)
    {
        balance = balance - amount;
    }

    protected int getId()
    {
        return id;
    }

    internal int getDatabaseId()
    {
        return databaseId;
    }

}
  
```



+ = public
 - = private
 # = protected
 ~ = package

Visibility – VB.Net

```

Class Account

  Private balance As Single = 0
  Public limit As Single = 0
  Protected id As Integer
  Friend databaseId As Integer

  Public Sub deposit(ByVal amount As Single)
    balance = balance + amount
  End Sub

  Private Sub withdraw(ByVal amount As Single)
    balance = balance - amount
  End Sub

  Protected Function getId() As Integer
    Return id
  End Function

  Friend Function getDatabaseId() As Integer
    Return databaseId
  End Function

End Class
  
```

```
short noOfPeople = Person.getNumberOfPeople();
Person p = Person.createPerson("Jason Gorman");
System.Diagnostics.Debug.Assert(Person.getNumberOfPeople()
== noOfPeople + 1);
```

Person
- <u>numberOfPeople</u> : int - name : string
+ <u>createPerson</u> (name : string) : Person + <u>getName</u> () : string + <u>getNumberOfPeople</u> () : int - Person(name : string)

Class & Instance Scope – C#

```
class Person
{
    private static int numberOfPeople = 0;
    private string name;

    private Person(string name)
    {
        this.name = name;
        numberOfPeople++;
    }

    public static Person createPerson(string name)
    {
        return new Person(name);
    }

    public string getName()
    {
        return this.name;
    }

    public static int getNumberOfPeople()
    {
        return numberOfPeople;
    }
}
```

```
Dim noOfPeople As Integer = Person.getNumberOfPeople()
Dim p As Person = Person.createPerson("Jason Gorman")
System.Diagnostics.Debug.Assert(Person.getNumberOfPeople() _
= noOfPeople + 1)
```

Person
- <u>numberOfPeople</u> : Integer - name : String
+ <u>createPerson</u> (name : String) : Person + <u>getName</u> () : String + <u>getNumberOfPeople</u> () : Integer - New(name : String)

Class & Instance Scope – VB.Net

```
Public Class Person

    Private Shared numberOfPeople As Integer = 0
    Private name As String

    Public Shared Function createPerson(ByVal name As String) As Person
        Return New Person(name)
    End Function

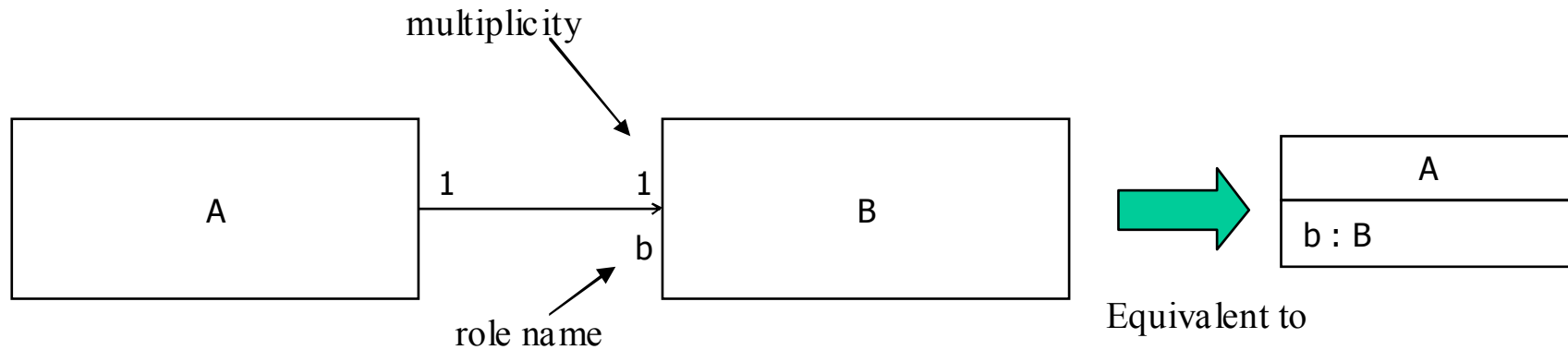
    Private Sub New(ByVal name As String)
        Me.name = name
        numberOfPeople = numberOfPeople + 1
    End Sub

    Public Function getName() As String
        Return name
    End Function

    Public Shared Function getNumberOfPeople() As Integer
        Return numberOfPeople
    End Function

End Class
```

Associations – C#



```
class A
{
    public B b = new B();
}

class B
{
}
```

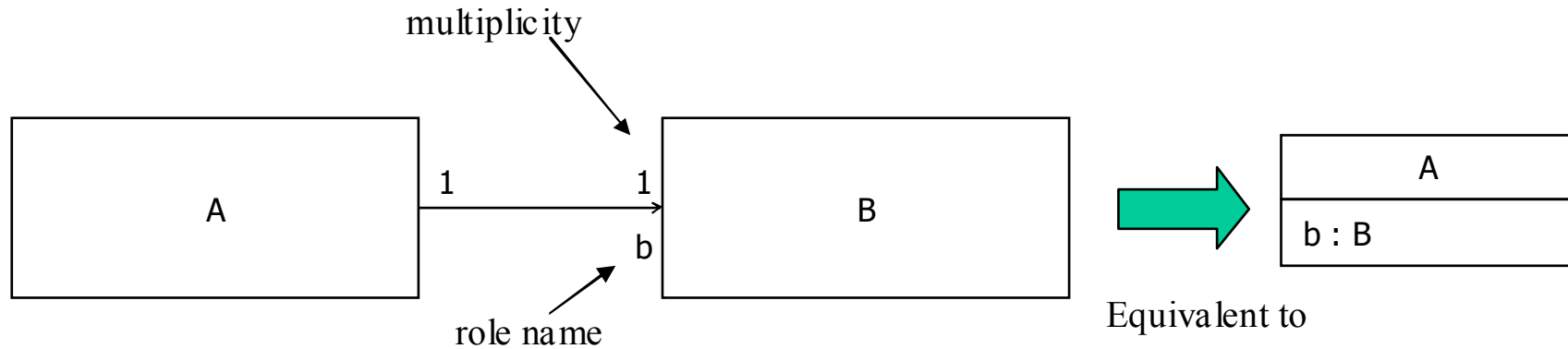
```
A a = new A();
B b = a.b;
```



The most advanced UML tutorial for C# developers

Get the UML for .NET Premium Package at
<http://www.parlezuml.com/tutorials/umlfordotnet.htm>

Associations – VB.Net

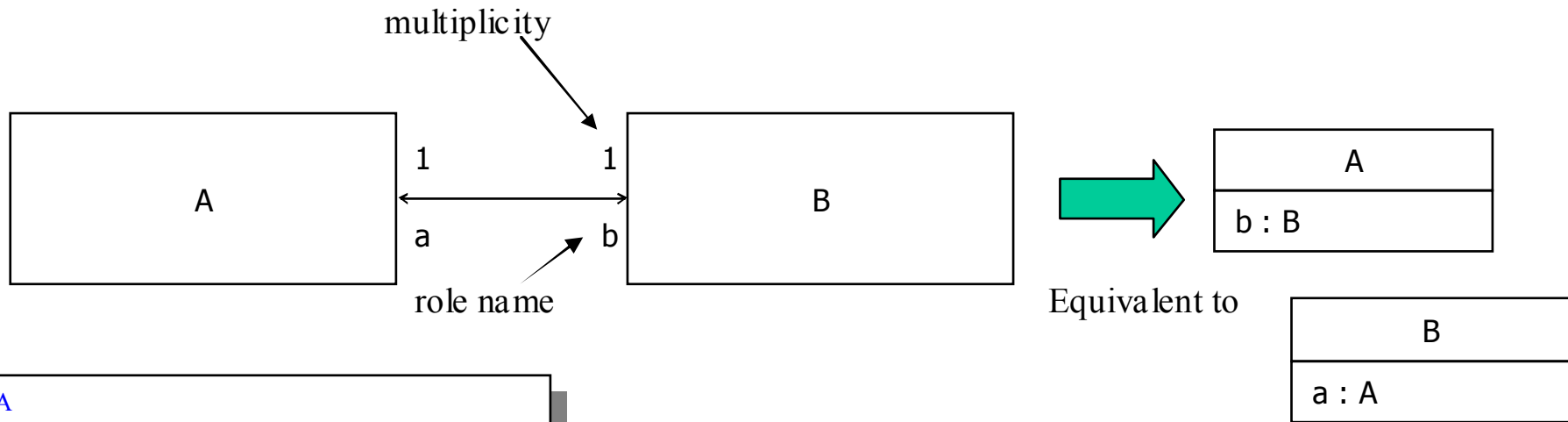


```
Class A
    Public b As New B()
End Class

Class B
End Class
```

```
Dim a As New A()
Dim b As B = a.b
```

Bi-directional Associations – C#

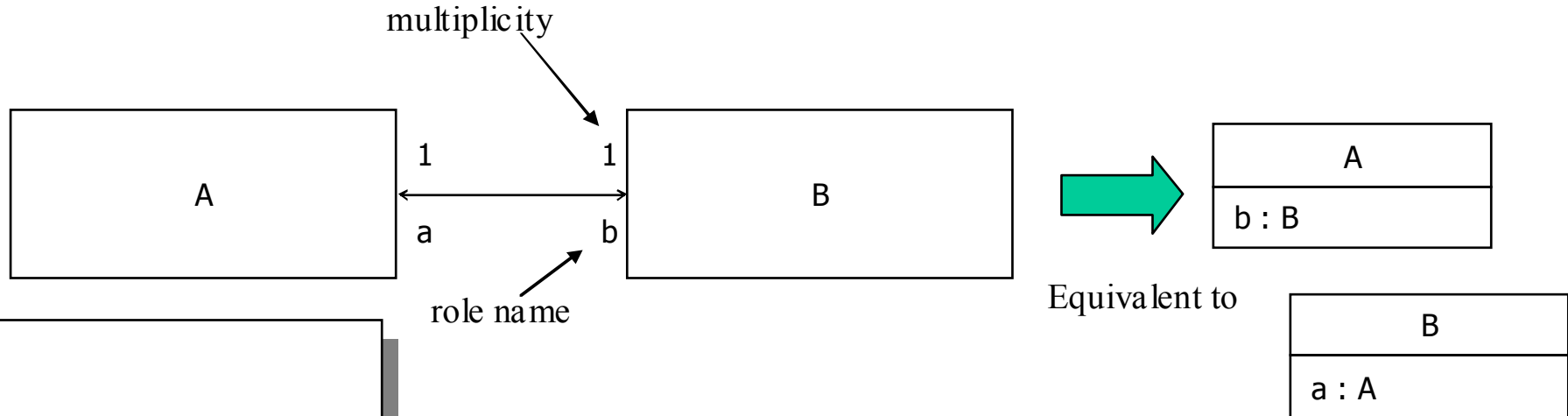


```
class A
{
    public B b;
    public A()
    {
        b = new B(this);
    }
}

class B
{
    public A a;
    public B(A a)
    {
        this.a = a;
    }
}
```

```
A a = new A();
B b = a.b;
A a1 = b.a;
System.Diagnostics.Debug.Assert(a == a1);
```

Bi-directional Associations – VB.Net



```

Class A

Public b As B()
Sub New()
    b = New B(Me)
End Sub
End Class

Class B

Public a As A
Public Sub New(ByRef a As A)
    Me.a = a
End Sub
End Class
    
```

```

Dim a As New A()
Dim b As B = a.b
Dim a1 As A = b.a
System.Diagnostics.Debug.Assert(a Is a1)
    
```

Association names & role defaults



Default role name = address
Default multiplicity = 1

```
class Person
{
    // association: Lives at
    public Address address;

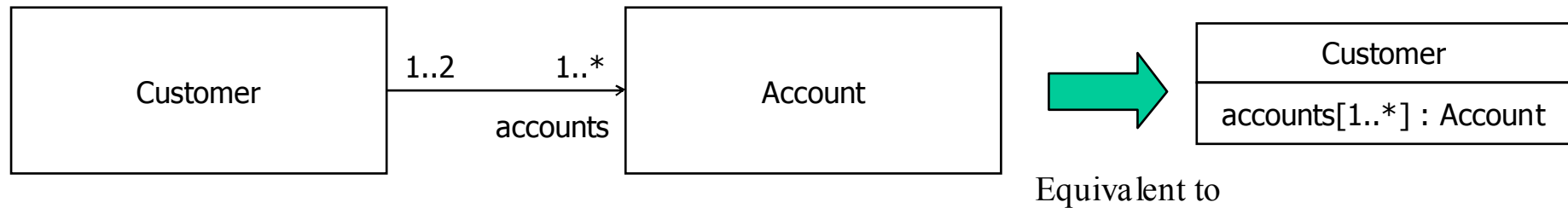
    public Person(Address address)
    {
        this.address = livesAt;
    }
}
```

```
Public Class Person
    ' association: Lives at
    Public address As Address

    Public Sub New(By Val address As address)
        Me.address = address
    End Sub

End Class
```

Multiplicity & Collections



```
class Customer
{
    // accounts[1..*] : Account
    public System.Collections.ArrayList accounts = new ArrayList();

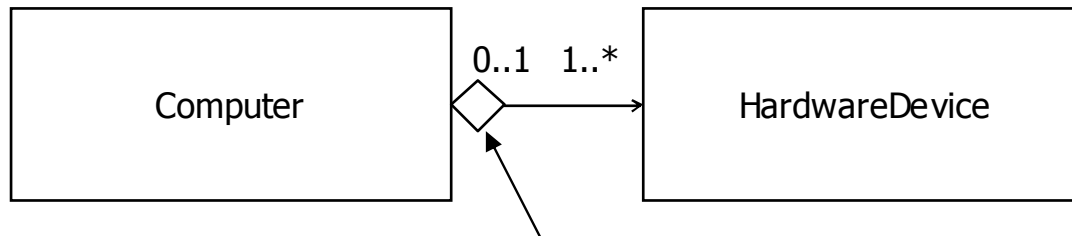
    public Customer()
    {
        Account defaultAccount = new Account();
        accounts.Add(defaultAccount);
    }
}
```

```
Class Customer
' accounts[1..*] : Account
Public accounts As New ArrayList()

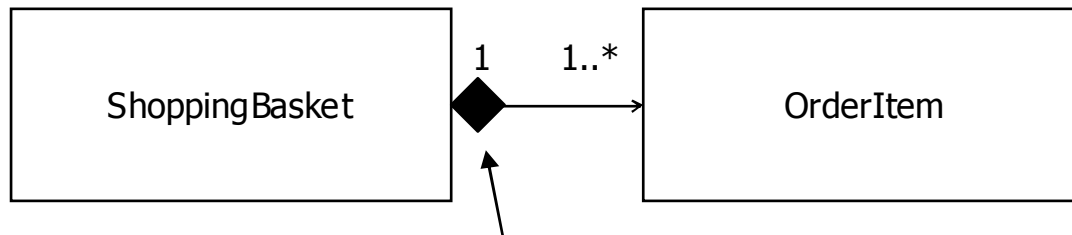
Public Sub New()
    Dim defaultAccount As New Account()
    accounts.Add(defaultAccount)
End Sub

End Class
```

Aggregation & Composition

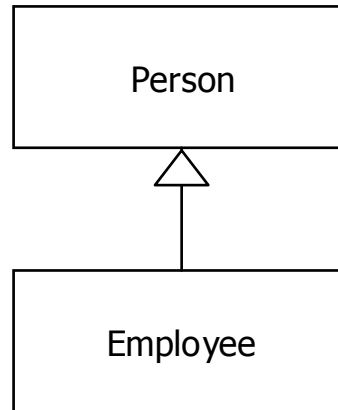


Aggregation – is made up of objects that can be shared or exchanged



Composition – is composed of objects that cannot be shared or exchanged and live only as long as the composite object

Generalization



```
class Person
{
}

class Employee : Person
{
}
```

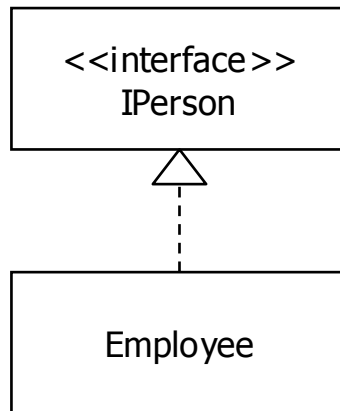
```
Class Person

End Class

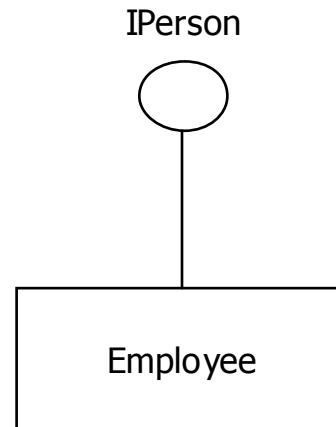
Class Employee
  Inherits Person

End Class
```

Realization



OR



```
interface IPerson
{
}

class Employee : IPerson
{
}
```

```
Interface IPerson

End Interface

Class Employee
  Implements IPerson

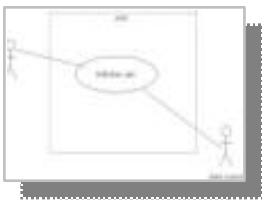
End Class
```

In the extended class diagrams tutorial...

- Overriding
- Abstract classes
- Dependencies
- Qualified associations
- Association classes
- More on associations, visibility & scope
- Information hiding

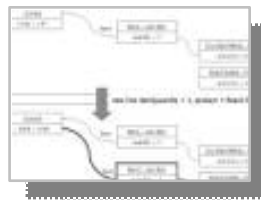


All with C# code examples



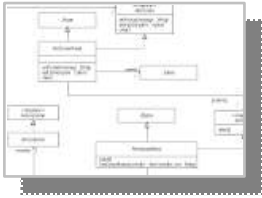
Use Case Diagrams

Model the users of the system and the goals they can achieve by using it



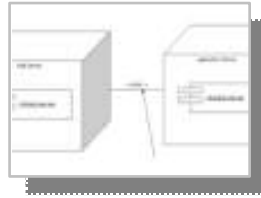
Object Diagrams & Filmstrips

Model snapshots of the running system and show how actions change object state



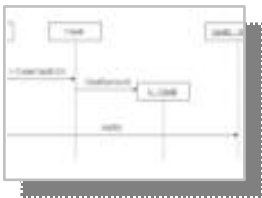
Class Diagrams

Model types of objects and the relationships between them.



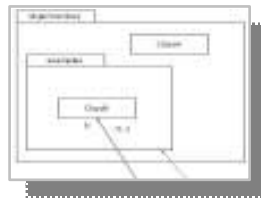
Implementation Diagrams

Model the physical components of a system and their deployment architecture



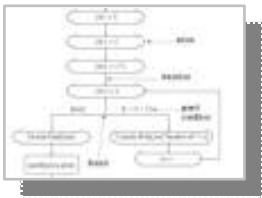
Sequence Diagrams

Model how objects interact to achieve functional goals



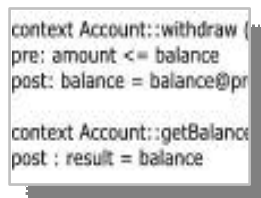
Packages & Model Management

Organise your logical and physical models with packages



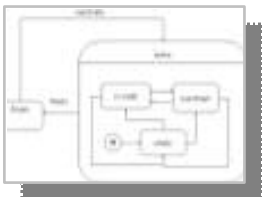
Activity Diagrams

Model the flow of use cases and single and multi-threaded code



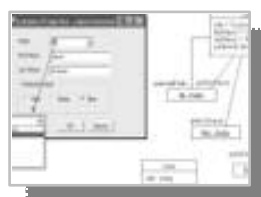
Object Constraint Language

Model business rules and create unambiguous specifications



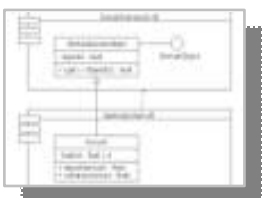
Statechart Diagrams

Model the behaviour of objects and event-driven applications



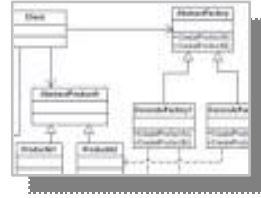
User Experience Modeling

Design user-centred systems with UML



Design Principles

Create well-designed software that's easier to change and reuse



Design Patterns

Apply proven solutions to common OO design problems



UML for .NET Premium Package

Available exclusively from www.parlezuml.com

