7.5.1 Projekte

Die vorgestellten drei Projekte nutzen in unterschiedlichen Varianten einfache und verschachtelte Collections.

7.5.1.1 Projekt 1

Das erste Projekt nutzt zwei Klassen mit erstaunlichen Ergebnissen und könnte die Basis für viele Projekte sein, bei denen eine Funktion *interaktiv* über eine geeignete Eingabe-Komponente (→ TextBox) bereitgestellt wird.

Es wird eine Zeichenkette – die eine mathematische Funktion repräsentiert – eingelesen sowie eine reelle Zahl als Argument x. Der Funktionswert f(x) wird berechnet und ausgegeben:

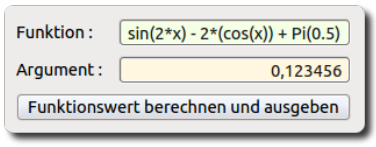

Abbildung 7.5.1.1.1: Programmoberfläche (Ausschnitt) für das Projekt 1

```
Public Function f(myFunction As String, x As Float) As Float
  Dim cContext As New Collection

  cContext["x"] = x
  Return Eval(myFunction, cContext)

End ' f(..)

' Kürzere Alternative:
' → Inline-Collection
' Public Function f(myFunction As String, x As Float) As Float
'   Return Eval(myFunction, ["x": x])
' End ' f(..)

Public Sub btnCalculate_Click()

  Print "f("; vbArgument.Value; ") = "; f(txbFunction.Text, vbArgument.Value)

End ' btnCalculate_Click()
```

Ausgabe:

```
f(0,123456) = -0,16957085489025
```

7.5.1.2 Projekt 2

So richtig praxisnah wird es, wenn man nicht die Funktionswerte *einer* Funktion sondern einer *Funktionsklasse* berechnen kann. Dafür muss die Funktion im Funktionsterm mindestens einen Parameter enthalten.

Das folgende Projekt setzt im Quelltext wieder auf die Verwendung der Klasse *Collection* und im Gegensatz zum Projekt 1 jetzt aber auf die Klasse *Expression*:

```
' Gambas class file

Public Sub Form_Open()
  vlbParamA.Type = vlbParamA.Number
  vlbParamA.SetFocus
  vlbParamB.Type = vlbParamB.Number
  vlb_X.Type = vlb_X.Number
  vlb_Y.Type = vlb_Y.Number
  vlb_Y.ReadOnly = True
' Initialisierung - Startwerte (Funktion, 2 Parameter):
  txbFKlasse.Text = "a*sin(x)-cos(b*x)+Pi(0.25)" ' Vorgabe-Funktion
  vlbParamA.Value = +0.125
```

```
  vlbParamB.Value = -2.5
  vlb_X.Value = 0.525
End ' Form_Open()

Public Sub btnShowResult_Click()
  ComputeY()
End ' btnComputeY_Click()

Public Sub btnClose_Click()
  FMain.Close
End ' btnClose_Click()

Private Function Compute(sExp As String, fParamA As Float, fParamB As Float, f_X As Float) As
Float
  Dim fFunktionswertY As Float
  Dim cEnvironment As New Collection
  Dim myExpression As New Expression

' ALTERNATIVE:
' cEnvironment.Add(fParamA, "a")
' cEnvironment.Add(fParamB, "b")
' cEnvironment.Add(f_X, "x")

  cEnvironment["a"] = fParamA ' Dem Symbol a wird der Wert vlbParameterA.Value zugewiesen
  cEnvironment["b"] = fParamB ' Dem Symbol b wird der Wert vlbParameterB.Value zugewiesen
  cEnvironment["x"] = f_X     ' Dem Symbol x wird der Wert vlbArgumentX.Value zugewiesen
  myExpression.Environment = cEnvironment

myExpression.Text = sExp

  Try fFunktionswertY = myExpression.Value

  If Error Then
     Message.Error("Fehlertext: " & gb.NewLine & Error.Text)
  Else
     Return fFunktionswertY
  Endif ' ERROR ?

End ' Function(..)

Public Sub ComputeY()
  vlb_Y.Value = Compute(txbFKlasse.Text, vlbParamA.Value, vlbParamB.Value, vlb_X.Value)
End ' btnComputeY_Click()
```

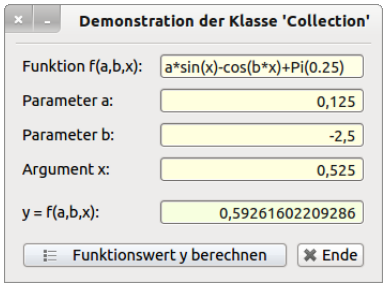

Abbildung 7.5.1.2.1: Programmoberfläche für das Projekt 2

7.5.1.3 Projekt 3

Ein Paradebeispiel für die Verwendung von Collections ist die Komponente *gb.settings*. Die gesamte Settings-Datei, die von einer Instanz der Settings-Klasse verwaltet wird, ist intern in einer Zwei-Ebenen-Hierarchie von Collections enthalten.

Es wird vereinfachend angenommen, dass die Konfigurationsdatei *.conf bereits vorhanden ist und nur ausgelesen sowie angezeigt werden soll. Es kann empfohlen werden, sich den adaptierten Quelltext im vorgestellten Projekt mehrfach genau durchzulesen, um die Wirkung der zwei relevanten Prozeduren zu ergründen.

Sie finden das vollständige Projekt als Archiv im Download-Bereich.

Abbildung 7.5.1.3.1: Programmoberfläche für das Projekt 3