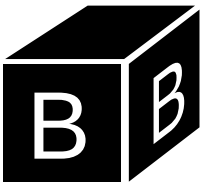


Chapter *23*

Inheritance and Polymorphism

This chapter describes six levels of abstraction.

Six abstraction processes

- Data abstraction, encompassing
  - Type abstraction, and
  - Structure abstraction
- Control abstraction, encompassing
  - Statement abstraction, and
  - Procedure abstraction
- Class abstraction
- Behavior abstraction

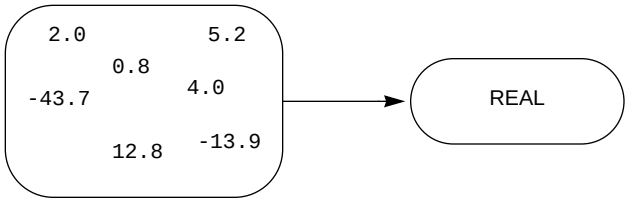


Figure 23.1
Type abstraction for type REAL.

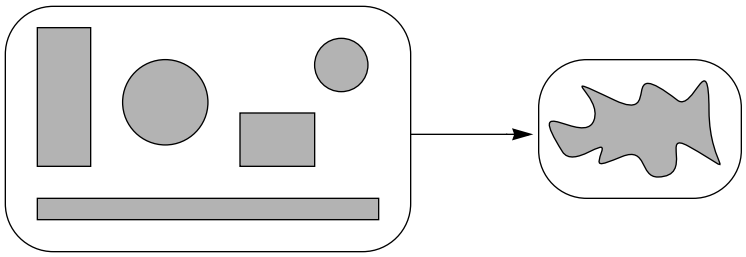


Figure 23.2
Structure abstraction to abstract from specific shapes of many different sizes to a single shape with a general size.

```
MODULE ...ShapeADT;
    TYPE
        Shape* = POINTER TO RECORD
            kind: INTEGER;
            length, width: REAL;
            diameter: REAL
        END


VAR
    myShape: ...ShapeADT.Shape


myShape.kind := 0;
myShape.length := 2.0;
myShape.width := 3.0
```

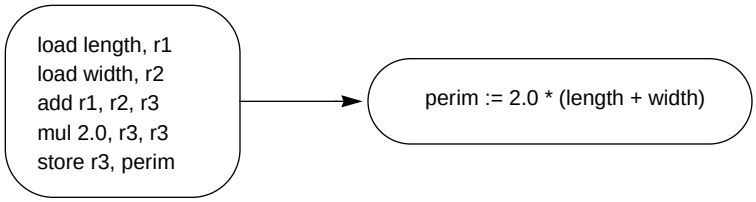


Figure 23.3
Statement abstraction for the assignment statement.

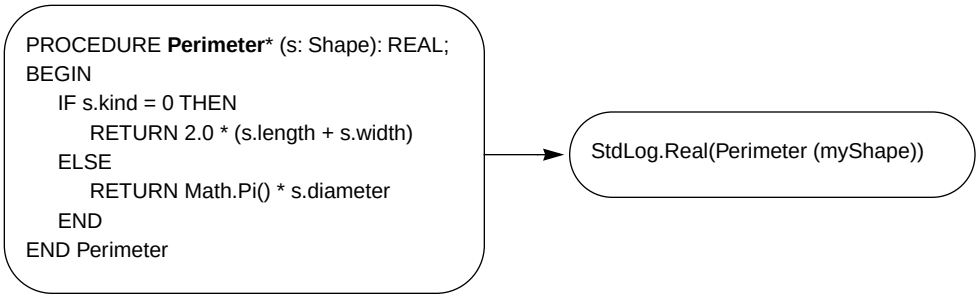


Figure 23.4
Procedure abstraction for the computation of the perimeter of a rectangle

```
StdLog.Real(Perimeter (myShape)); StdLog.Ln;
StdLog.Real(Perimeter (yourShape)); StdLog.Ln
```

```
DEFINITION ...ShapeADT;
   TYPE
      Shape = POINTER TO RECORD END;
   PROCEDURE Area (s: Shape): REAL;
   PROCEDURE Perimeter (s: Shape): REAL;


DEFINITION ...ShapeObj;
   TYPE
      Shape = POINTER TO RECORD
         (s: Shape) Area (): REAL, NEW;
         (s: Shape) Perimeter (): REAL, NEW
      END;


MODULE ...ShapeObj;
   TYPE
      Shape* = POINTER TO RECORD
         kind: INTEGER;
         length, width: REAL;
         diameter: REAL
      END
```

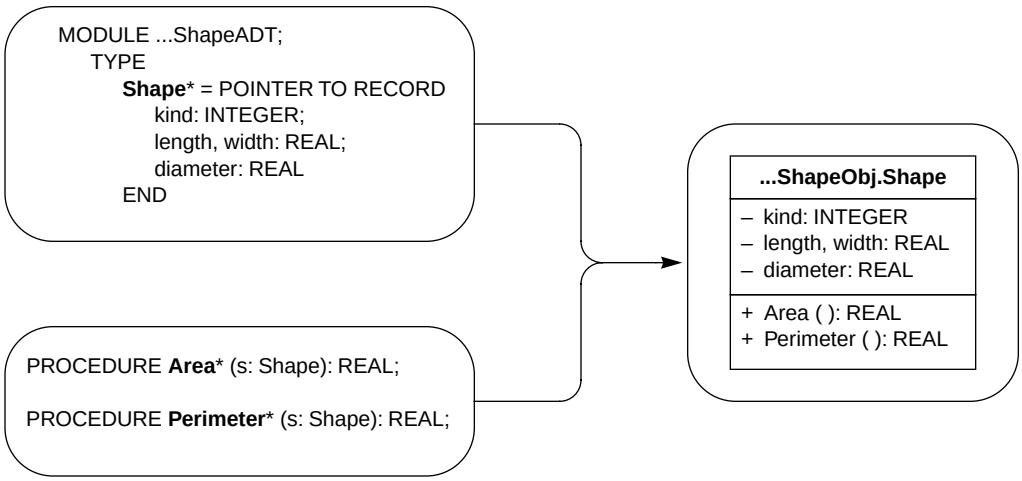


Figure 23.5
Class abstraction that combines the structure abstraction of Figure 23.2 with the procedure abstraction of Figure 23.4.

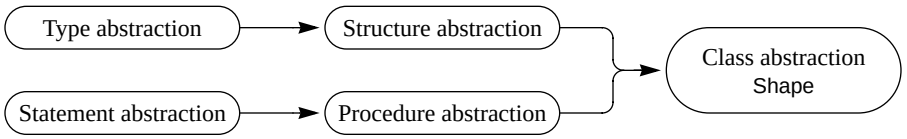
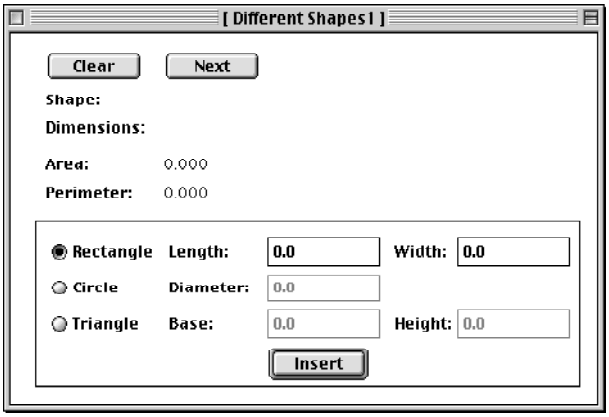


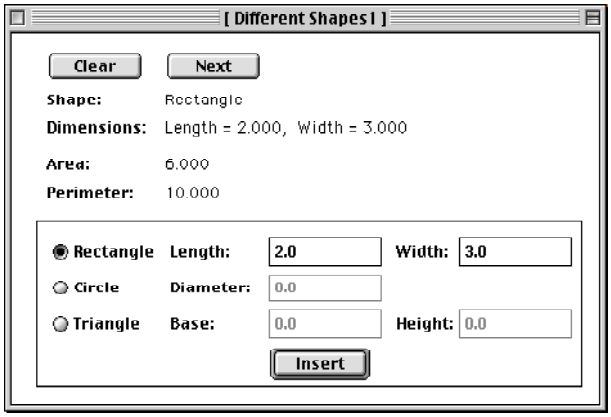
Figure 23.6
Using class abstraction without behavior abstraction to process data for several different kinds of shapes.

```
DEFINITION PboxShapeObj;

    TYPE
        Shape = POINTER TO RECORD
            (s: Shape) GetIDString (OUT str: ARRAY OF CHAR), NEW;
            (s: Shape) GetDimensionString (OUT str: ARRAY OF CHAR), NEW;
            (s: Shape) Area (): REAL, NEW;
            (s: Shape) Perimeter (): REAL, NEW;
            (s: Shape) SetRectangleState (length, width: REAL), NEW;
            (s: Shape) SetCircleState (diameter: REAL), NEW
        END;

END PboxShapeObj.
```

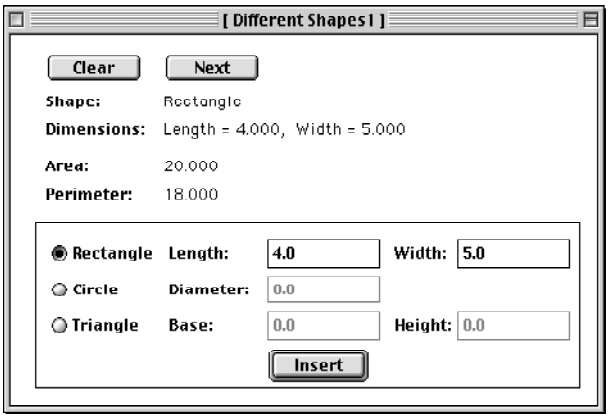
Figure 23.7
The interface for a Shape class without behavior abstraction.



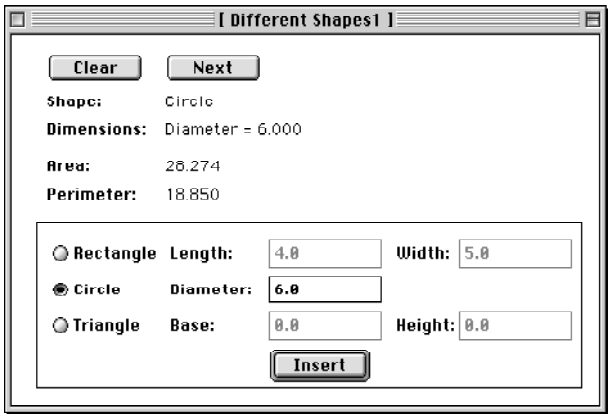
(a) Initial.



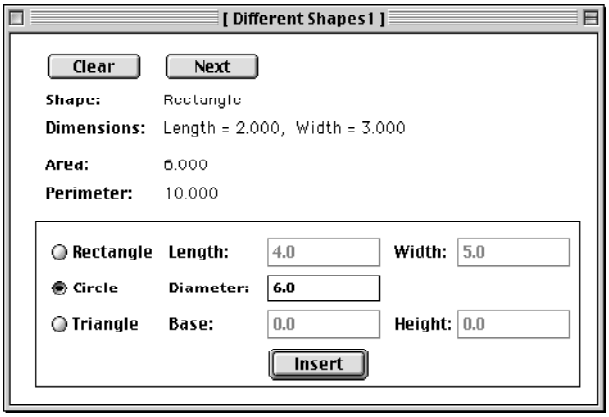
(b) Insert a rectangle.



(c) Insert a rectangle.



(d) Insert a circle.



(e) Press Next.

Figure 23.8
A sequence of screen shots for execution of a program to process shapes.

```
MODULE Pbox23A;
   IMPORT Dialog, C := PboxCListADT, S := PboxShapeObj;

   VAR
      d*: RECORD
         idString-, dimensionString- : ARRAY 64 OF CHAR;
         area-, perimeter-: REAL;
         shapeNumber*: INTEGER;
         length*, width*: REAL; (* for rectangle *)
         diameter*: REAL; (* for circle *)
         base*, height*: REAL (* for triangle *)
      END;
      cList: C.CList;
```

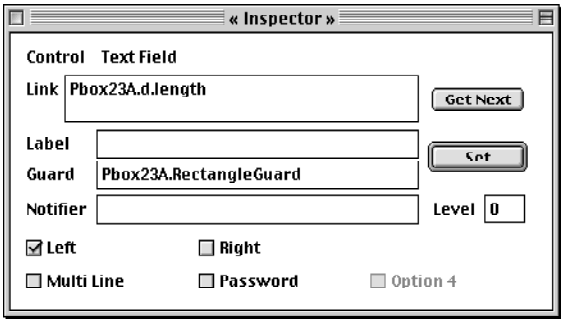
Figure 23.9
A program that produces the sequence of screen shots in Figure 23.8.

```
PROCEDURE ClearDialog;
BEGIN
    d.idString := ""; d.dimensionString := "";
    d.area := 0.0; d.perimeter := 0.0;
    d.length := 0.0; d.width := 0.0;
    d.diameter := 0.0;
    d.base := 0.0; d.height := 0.0
END ClearDialog;

PROCEDURE SetDialog (s: S.Shape);
BEGIN
    s.GetIDString(d.idString);
    s.GetDimensionString(d.dimensionString);
    d.area := s.Area();
    d.perimeter := s.Perimeter()
END SetDialog;
```

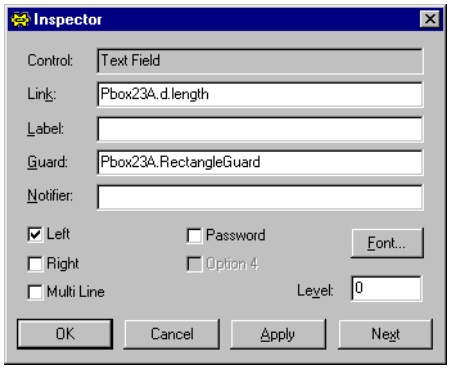
```
PROCEDURE Clear*;
BEGIN
    ClearDialog;
    C.Clear(cList);
    Dialog.Update(d)
END Clear;

PROCEDURE Next*;
    VAR
        shape: S.Shape;
BEGIN
    IF ~C.Empty(cList) THEN
        C.GoNext(cList);
        shape := C.NodeContent(cList) (S.Shape);
        SetDialog(shape);
        Dialog.Update(d)
    END
END Next;
```

```
PROCEDURE Insert*;
    VAR
        shape: S.Shape;
BEGIN
    NEW(shape);
    CASE d.shapeNumber OF
    0:
        shape.SetRectangleState(MAX(0.0, d.length), MAX(0.0, d.width)) |
    1:
        shape.SetCircleState(MAX(0.0, d.diameter)) |
    2:
        (* Problem for the student *)
        HALT(100)
    END;
    C.Insert(cList, shape);
    SetDialog(shape);
    Dialog.Update(d)
END Insert;
```

```
    PROCEDURE RectangleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 0
    END RectangleGuard;

    PROCEDURE CircleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 1
    END CircleGuard;

    PROCEDURE TriangleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 2
    END TriangleGuard;

BEGIN
    Clear
END Pbox23A.
```



(a) MacOS.



(b) MSWindows.

Figure 23.10
Links that must be entered in the Inspector to enable the guards for dimming controls in the dialog box.

```
DEFINITION Dialog;

    TYPE
        Par = RECORD
            disabled: BOOLEAN;
            checked: BOOLEAN;
            undef: BOOLEAN;
            readOnly: BOOLEAN;
            label: String
        END;
```

Figure 23.11
A partial listing of the documentation for the Dialog module.

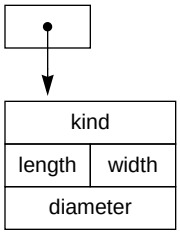


Figure 23.12
The structure of type Shape in Figure 23.13.

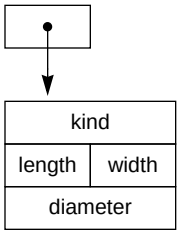


Figure 23.12
The structure of type Shape in Figure 23.13.

```
MODULE  PboxShapeObj;
   IMPORT PboxStrings, Math;

   TYPE
      Shape* = POINTER TO RECORD
         kind: INTEGER;
         length, width: REAL;
         diameter: REAL
      END;
```

Figure 23.13
Implementation of the shape class with class abstraction only. Figure 23.7 shows the interface for this class.

```
PROCEDURE (s: Shape) GetIDString* (OUT str: ARRAY OF CHAR), NEW;
BEGIN
    CASE s.kind OF
    0:
        str := "Rectangle" |
    1:
        str := "Circle"
    END
END GetIDString;
```

```
PROCEDURE (s: Shape) GetDimensionString* (OUT str: ARRAY OF CHAR), NEW;
    VAR
        temp: ARRAY 16 OF CHAR;
BEGIN
    CASE s.kind OF
    0:
        PboxStrings.RealToString(s.length, 1, 3, temp);
        str := "Length = " + temp + ",  ";
        PboxStrings.RealToString(s.width, 1, 3, temp);
        str := str + "Width = " + temp |
    1:
        PboxStrings.RealToString(s.diameter, 1, 3, temp);
        str := "Diameter = " + temp
    END
END GetDimensionString;
```

```
PROCEDURE (s: Shape) Area* (): REAL, NEW;
BEGIN
   CASE s.kind OF
   0:
      RETURN s.length * s.width |
   1:
      RETURN Math.Pi() * s.diameter * s.diameter / 4.0
   END
END Area;

PROCEDURE (s: Shape) Perimeter* (): REAL, NEW;
BEGIN
   CASE s.kind OF
   0:
      RETURN 2.0 * (s.length + s.width) |
   1:
      RETURN Math.Pi() * s.diameter
   END
END Perimeter;
```

```
    PROCEDURE (s: Shape) SetRectangleState* (length, width: REAL), NEW;
    BEGIN
        ASSERT((length >= 0.0) & (width >= 0.0), 20);
        s.kind := 0;
        s.length := length;
        s.width := width
    END SetRectangleState;

    PROCEDURE (s: Shape) SetCircleState* (diameter: REAL), NEW;
    BEGIN
        ASSERT(diameter >= 0.0, 20);
        s.kind := 1;
        s.diameter := diameter
    END SetCircleState;

END PboxShapeObj.
```

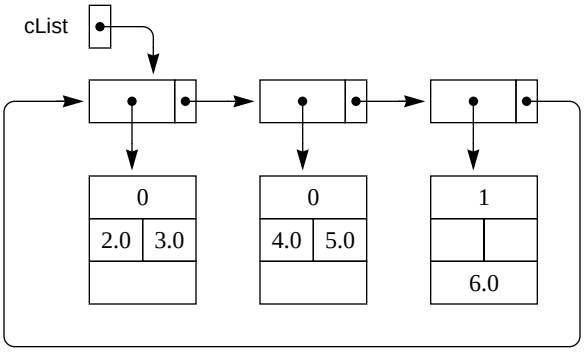


Figure 23.14
The data structure that corresponds to the screen shot of Figure 23.8(e) with the shape of PboxShapeObj.

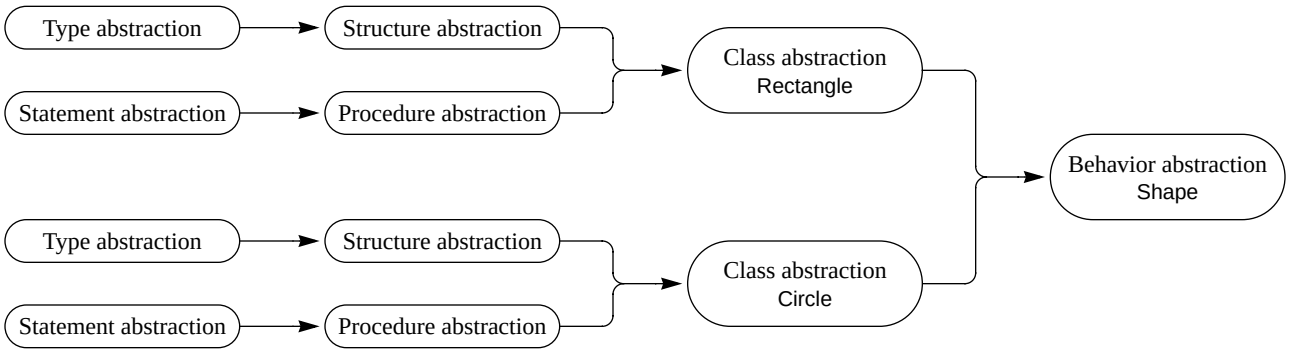


Figure 23.15
Behavior abstraction that combines class abstraction for two different classes into a single abstract class.

```
DEFINITION PboxShapeAbs;

   TYPE
      Shape = POINTER TO ABSTRACT RECORD
         (s: Shape) GetIDString (OUT str: ARRAY OF CHAR), NEW, ABSTRACT;
         (s: Shape) GetDimensionString (OUT str: ARRAY OF CHAR), NEW, ABSTRACT;
         (s: Shape) Area (): REAL, NEW, ABSTRACT;
         (s: Shape) Perimeter (): REAL, NEW, ABSTRACT
      END;

      Rectangle = POINTER TO RECORD (Shape)
         (r: Rectangle) GetIDString (OUT str: ARRAY OF CHAR);
         (r: Rectangle) GetDimensionString (OUT str: ARRAY OF CHAR);
         (r: Rectangle) Area (): REAL;
         (r: Rectangle) Perimeter (): REAL;
         (r: Rectangle) SetState (length, width: REAL), NEW
      END;

      Circle = POINTER TO RECORD (Shape)
         (c: Circle) GetIDString (OUT str: ARRAY OF CHAR);
         (c: Circle) GetDimensionString (OUT str: ARRAY OF CHAR);
         (c: Circle) Area (): REAL;
         (c: Circle) Perimeter (): REAL;
         (c: Circle) SetState (diameter: REAL), NEW
      END;

END PboxShapeAbs.
```

Figure 23.16
The interface for a general geometric shape. that uses behavior abstraction.

```
MODULE Pbox23B;
   IMPORT Dialog, C := PboxCListADT, S := PboxShapeAbs;

   VAR
      d*: RECORD
         idString-, dimensionString- : ARRAY 64 OF CHAR;
         area-, perimeter-: REAL;
         shapeNumber*: INTEGER;
         length*, width*: REAL; (* for rectangle *)
         diameter*: REAL; (* for circle *)
         base*, height*: REAL (* for triangle *)
      END;
      cList: C.CList;
```

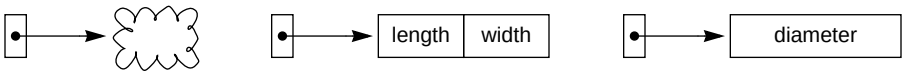
Figure 23.17
A program that produces the same output as the one in Figure 23.9 but that uses behavior abstraction.

```
PROCEDURE ClearDialog;
BEGIN
    d.idString := ""; d.dimensionString := "";
    d.area := 0.0; d.perimeter := 0.0;
    d.length := 0.0; d.width := 0.0;
    d.diameter := 0.0;
    d.base := 0.0; d.height := 0.0
END ClearDialog;

PROCEDURE SetDialog (s: S.Shape);
BEGIN
    s.GetIDString(d.idString);
    s.GetDimensionString(d.dimensionString);
    d.area := s.Area();
    d.perimeter := s.Perimeter()
END SetDialog;
```

```
PROCEDURE Clear*;
BEGIN
   ClearDialog;
   C.Clear(cList);
   Dialog.Update(d)
END Clear;

PROCEDURE Next*;
   VAR
      shape: S.Shape;
BEGIN
   IF ~C.Empty(cList) THEN
      C.GoNext(cList);
      shape := C.NodeContent(cList) (S.Shape);
      SetDialog(shape);
      Dialog.Update(d)
   END
END Next;
```

```
PROCEDURE Insert*;
   VAR
      rectangle: S.Rectangle;
      circle: S.Circle;
BEGIN
   CASE d.shapeNumber OF
   0:
      NEW(rectangle);
      rectangle.SetState(MAX(0.0, d.length), MAX(0.0, d.width));
      C.Insert(cList, rectangle);
      SetDialog(rectangle) |
   1:
      NEW(circle);
      circle.SetState(MAX(0.0, d.diameter));
      C.Insert(cList, circle);
      SetDialog(circle) |
   2:
      (* Problem for the student *)
   END;
   Dialog.Update(d)
END Insert;
```

```
    PROCEDURE RectangleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 0
    END RectangleGuard;

    PROCEDURE CircleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 1
    END CircleGuard;

    PROCEDURE TriangleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 2
    END TriangleGuard;

BEGIN
    Clear
END Pbox23B.
```



(a) Shape. **(b)** Rectangle. **(c)** Circle.

Figure 23.18
The structure of type Shape, Rectangle, and Circle in Figure 23.19.

```
MODULE PboxShapeAbs;
   IMPORT PboxStrings, Math;

   TYPE
      Shape* = POINTER TO ABSTRACT RECORD END;

      Rectangle* = POINTER TO RECORD (Shape)
         length, width: REAL
      END;

      Circle* = POINTER TO RECORD (Shape)
         diameter: REAL
      END;
```

Figure 23.19
Implementation of the abstract class Shape whose interface is in Figure 23.16.

```
(* ---------------------- *)
    PROCEDURE (s: Shape) GetIDString* (OUT str: ARRAY OF CHAR), NEW, ABSTRACT;

    PROCEDURE (r: Rectangle) GetIDString* (OUT str: ARRAY OF CHAR);
    BEGIN
        str := "Rectangle"
    END GetIDString;

    PROCEDURE (c: Circle) GetIDString* (OUT str: ARRAY OF CHAR);
    BEGIN
        str := "Circle"
    END GetIDString;
```

```
(* --------------------- *)
    PROCEDURE (s: Shape) GetDimensionString* (OUT str: ARRAY OF CHAR), NEW, ABSTRACT;

    PROCEDURE (r: Rectangle) GetDimensionString* (OUT str: ARRAY OF CHAR);
        VAR
            temp: ARRAY 16 OF CHAR;
    BEGIN
        PboxStrings.RealToString(r.length, 1, 3, temp);
        str := "Length = " + temp + ",  ";
        PboxStrings.RealToString(r.width, 1, 3, temp);
        str := str + "Width = " + temp
    END GetDimensionString;

    PROCEDURE (c: Circle) GetDimensionString* (OUT str: ARRAY OF CHAR);
        VAR
            temp: ARRAY 16 OF CHAR;
    BEGIN
        PboxStrings.RealToString(c.diameter, 1, 4, temp);
        str := "Diameter = " + temp
    END GetDimensionString;
```

```
(* ---------------------- *)
    PROCEDURE (s: Shape) Area* (): REAL, NEW, ABSTRACT;

    PROCEDURE (r: Rectangle) Area* (): REAL;
    BEGIN
        RETURN r.length * r.width
    END Area;

    PROCEDURE (c: Circle) Area* (): REAL;
    BEGIN
        RETURN Math.Pi() * c.diameter * c.diameter / 4.0
    END Area;
```

```
(* ---------------------- *)
    PROCEDURE (s: Shape) Perimeter* (): REAL, NEW, ABSTRACT;

    PROCEDURE (r: Rectangle) Perimeter* (): REAL;
    BEGIN
        RETURN 2.0 * (r.length + r.width)
    END Perimeter;

    PROCEDURE (c: Circle) Perimeter* (): REAL;
    BEGIN
        RETURN Math.Pi() * c.diameter
    END Perimeter;
```

```
(* ---------------------- *)
    PROCEDURE (r: Rectangle) SetState* (length, width: REAL), NEW;
    BEGIN
        ASSERT((length >= 0.0) & (width >= 0.0), 20);
        r.length := length;
        r.width := width
    END SetState;

    PROCEDURE (c: Circle) SetState* (diameter: REAL), NEW;
    BEGIN
        ASSERT(diameter >= 0.0, 20);
        c.diameter := diameter
    END SetState;

END PboxShapeAbs.
```

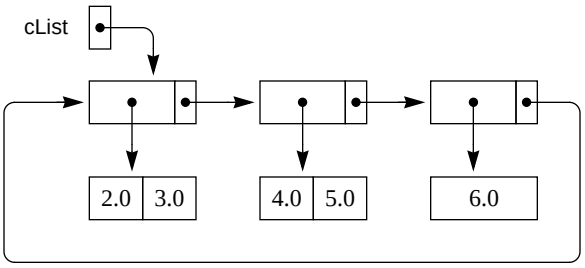


Figure 23.20
The data structure that corresponds to Figure 23.14, but with the abstract shape of PboxShapeABS.

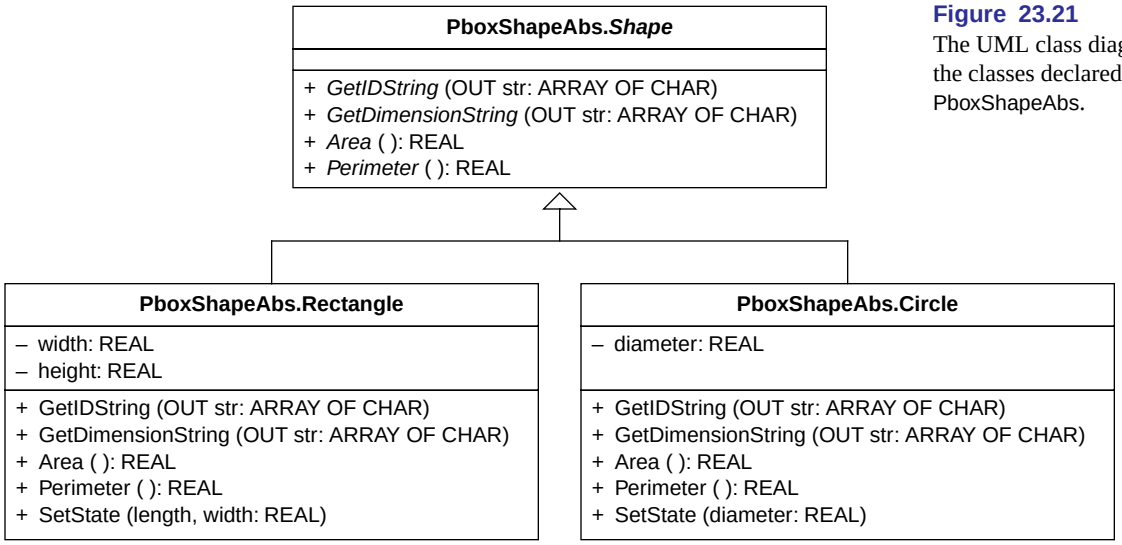


Figure 23.21
The UML class diagram for the classes declared in PboxShapeAbs.

(a) The full version of the class diagram.

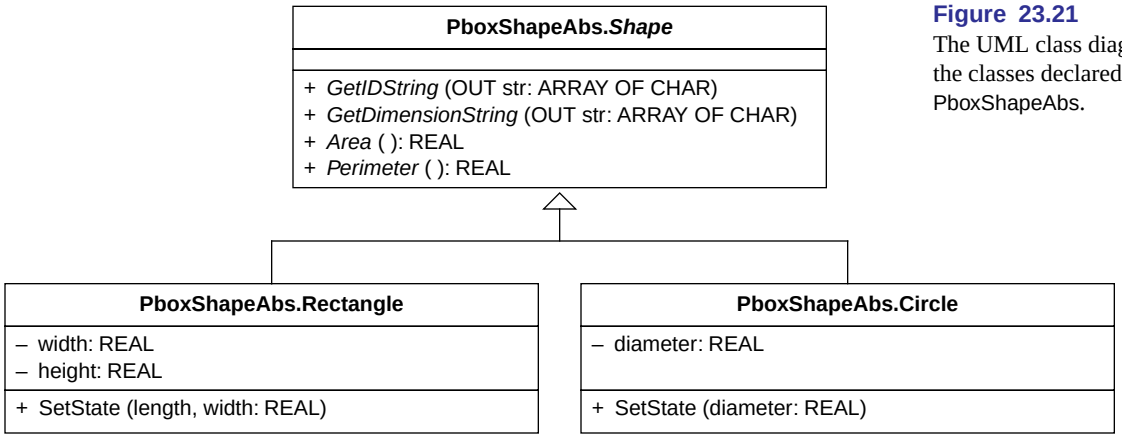


Figure 23.21
The UML class diagram for the classes declared in PboxShapeAbs.

(b) The abbreviated version of the class diagram.

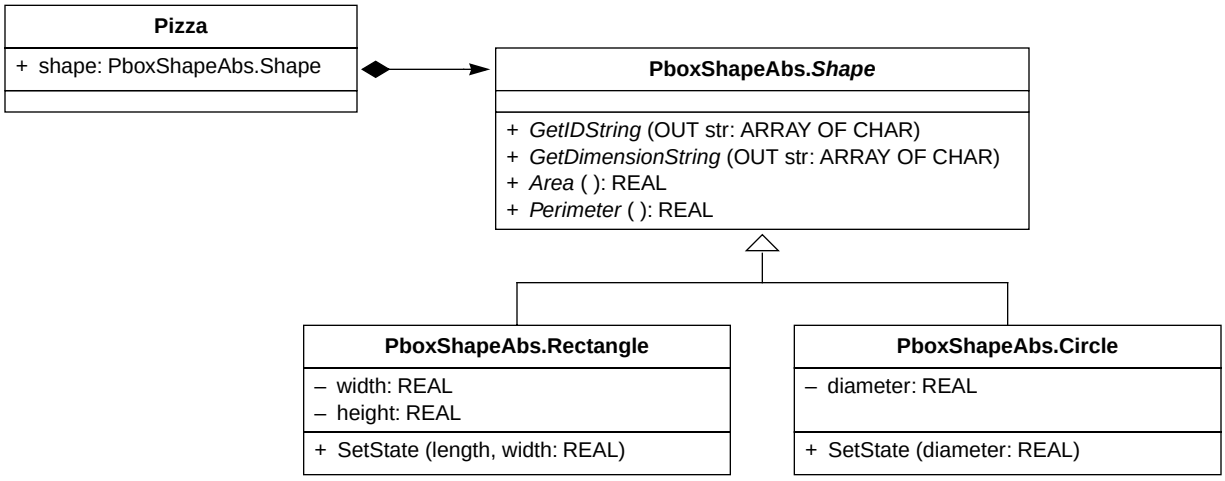


Figure 23.22
The UML class diagram for a Pizza class that has a Shape.

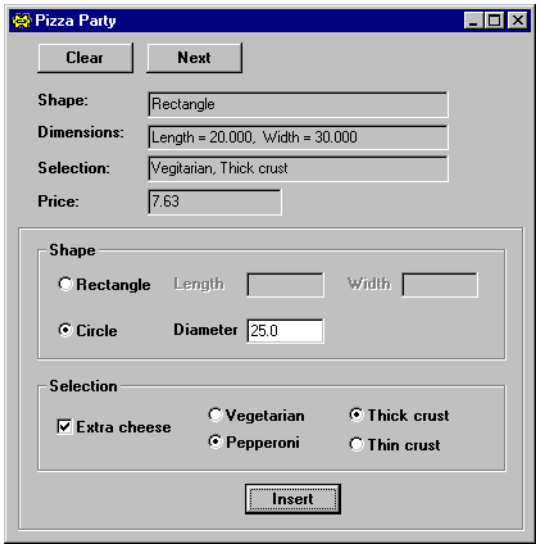


Figure 23.23
The dialog box for a program that uses the Pizza class.

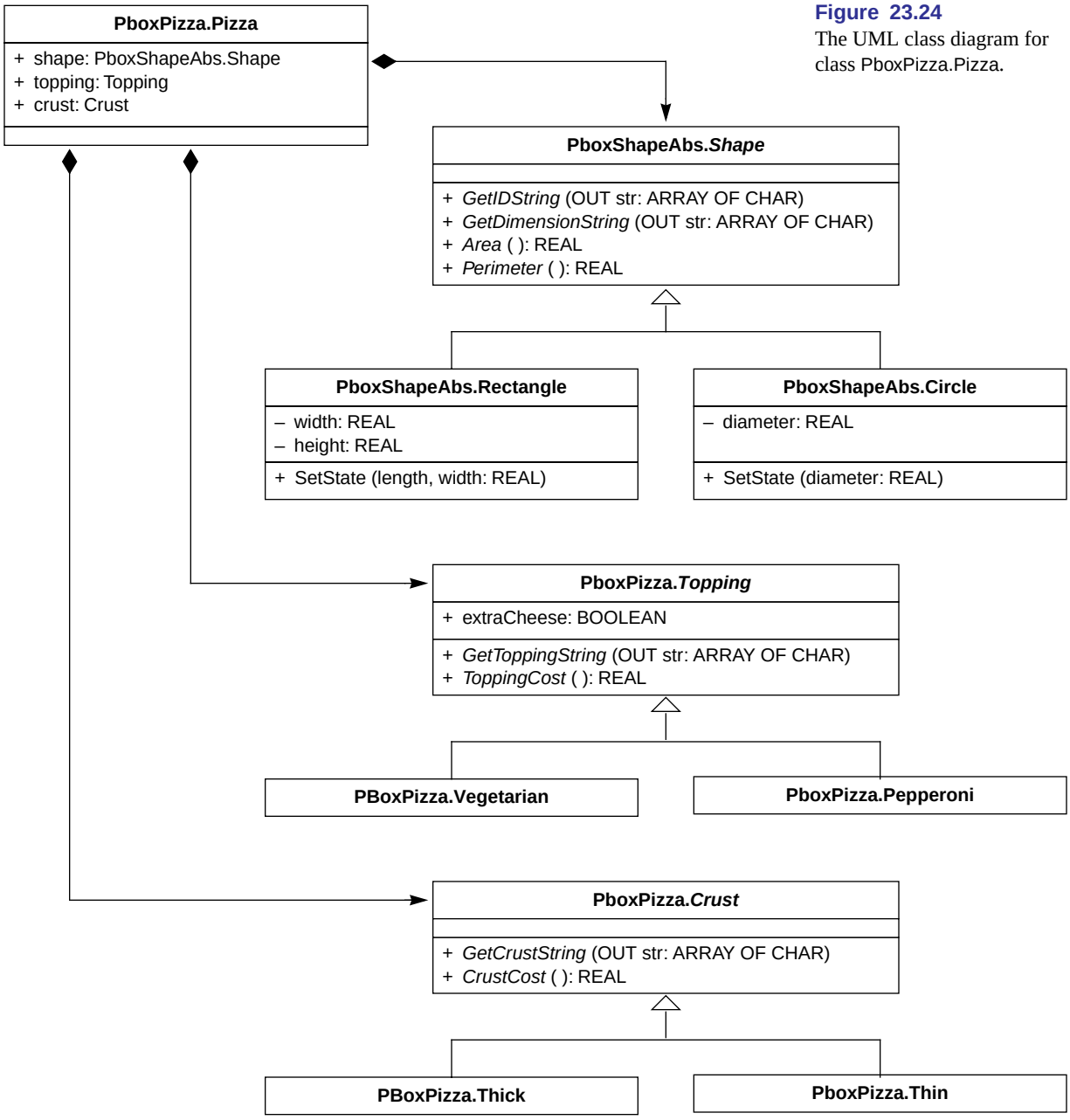


Figure 23.24
The UML class diagram for class PboxPizza.Pizza.

```
MODULE PboxPizza;
   IMPORT PboxShapeAbs;

   TYPE
      Topping* = POINTER TO ABSTRACT RECORD
         (* Problem for the student. *)
      END;

      (* Topping subclasses, Problem for the student. *)
      (* Crust class and subclasses, Problem for the student. *)

      Pizza* = POINTER TO RECORD
         shape*: PboxShapeAbs.Shape;
         topping*: Topping;
         (* Problem for the student *)
      END;
```

Figure 23.25
Implementation of the Pizza class whose UML class diagram is in Figure 23.24.

```
(* --------------------- *)
   PROCEDURE (t: Topping) GetToppingString* (OUT str: ARRAY OF CHAR), NEW, ABSTRACT;

   (* GetToppingString, Problem for the student. *)

(* --------------------- *)
   PROCEDURE (t: Topping) ToppingCost* (): REAL, NEW, ABSTRACT;

   (* ToppingCost, Problem for the student. *)

(* --------------------- *)

   (* GetCrustString, Problem for the student *)

(* --------------------- *)

   (* CrustCost, Problem for the student *)

END PboxPizza.
```

```
MODULE Pbox23C;
   IMPORT Dialog, C := PboxCListADT, S := PboxShapeAbs, P := PboxPizza;

   CONST
      basePrice = 2.50;
      tax = 0.09;

   VAR
      d*: RECORD
         shapeString-, dimensionString-, selectionString-: ARRAY 64 OF CHAR;
         price-: REAL;
         shapeNumber*: INTEGER;
         length*, width*: REAL; (* for rectangle *)
         diameter*: REAL; (* for circle *)
         extraCheese*: BOOLEAN;
         toppingNumber*, crustNumber*: INTEGER
      END;
      cList: C.CList;
```

Figure 23.26
The program for the dialog box of Figure 23.23.

```
PROCEDURE ClearDialog;
BEGIN
    d.shapeString := ""; d.dimensionString := ""; d.selectionString := "";
    d.price := 0.0;
    d.shapeNumber := 0;
    d.length := 0.0; d.width := 0.0;
    d.diameter := 0.0;
    d.extraCheese := FALSE;
    d.toppingNumber := 0; d.crustNumber := 0
END ClearDialog;

PROCEDURE SetDialog (pz: P.Pizza);
    VAR
        tempStr: ARRAY 32 OF CHAR;
BEGIN
    pz.shape.GetIDString(d.shapeString);
    pz.shape.GetDimensionString(d.dimensionString);
    (* Problem for the student *)
END SetDialog;
```

```
PROCEDURE Clear*;
BEGIN
   ClearDialog;
   C.Clear(cList);
   Dialog.Update(d)
END Clear;

PROCEDURE Next*;
   VAR
      pizza: P.Pizza;
BEGIN
   IF ~C.Empty(cList) THEN
      C.GoNext(cList);
      pizza := C.NodeContent(cList) (P.Pizza);
      SetDialog(pizza);
      Dialog.Update(d)
   END
END Next;
```

```
PROCEDURE Insert*;
    VAR
        pizza: P.Pizza;
        rectangle: S.Rectangle;
        circle: S.Circle;
        (* Problem for the student. *)
BEGIN
    NEW(pizza);
    CASE d.shapeNumber OF
    0:
        NEW(rectangle);
        rectangle.SetState(MAX(0.0, d.length), MAX(0.0, d.width));
        pizza.shape := rectangle |
    1:
        NEW(circle);
        circle.SetState(MAX(0.0, d.diameter));
        pizza.shape := circle
    END;
    (* CASE d.toppingNumber, Problem for the student *)
    (* d.extraCheese, Problem for the student *)
    (* CASE d.crustNumber, Problem for the student *)
    C.Insert(cList, pizza);
    SetDialog(pizza);
    Dialog.Update(d)
END Insert;
```

```
    PROCEDURE RectangleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 0
    END RectangleGuard;

    PROCEDURE CircleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 1
    END CircleGuard;

    PROCEDURE TriangleGuard* (VAR par: Dialog.Par);
    BEGIN
        par.disabled := d.shapeNumber # 1
    END TriangleGuard;

BEGIN
    Clear
END Pbox23C.
```

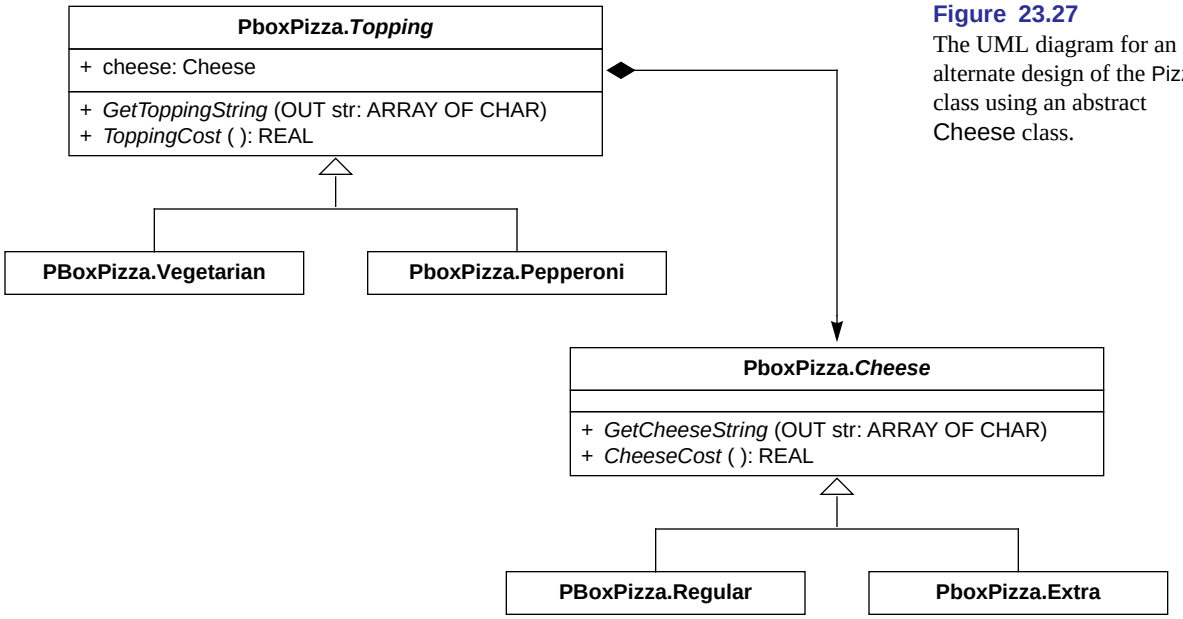


Figure 23.27
The UML diagram for an alternate design of the Pizza class using an abstract Cheese class.

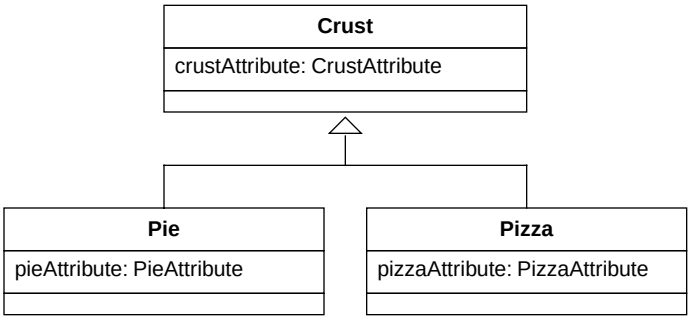


Figure 23.28
A possible design with a different relationship between Pizza and Crust.

A curious restriction on abstract objects and methods are the following two rules.

Restrictions on abstract objects and methods

- You cannot instantiate an abstract object with NEW.
- You cannot implement an abstract method with BEGIN..END.