Chapter 6

Selection Statements

OBJECTIVES

After you have read and studied this chapter, you should be able to

- Implement selection control in a program using `if` statements.
- Implement selection control in a program using `switch` statements.
- Write boolean expressions using relational and boolean operators.
- Evaluate given boolean expressions correctly.
- Nest an `if` statement inside another `if` statement's then or else part correctly.
- Choose the appropriate selection control statement for a given task.

FIGURE 6.1 Mapping of the sample **if** statement to the general format.

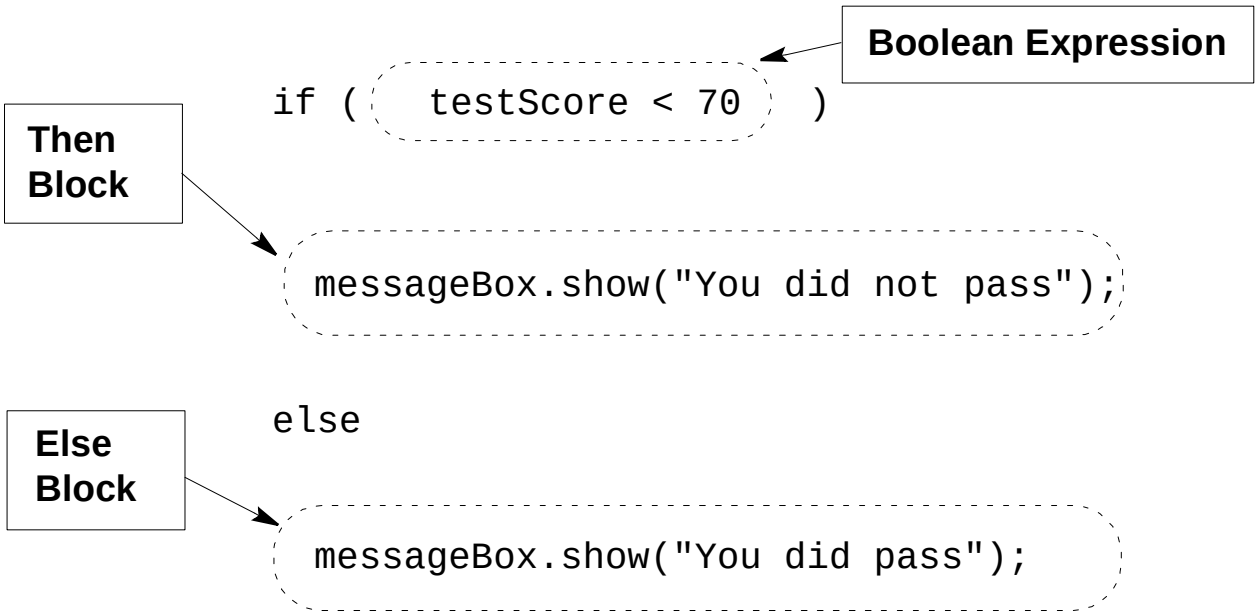


FIGURE 6.2 The diagram showing the control flow of the sample **if** statement.

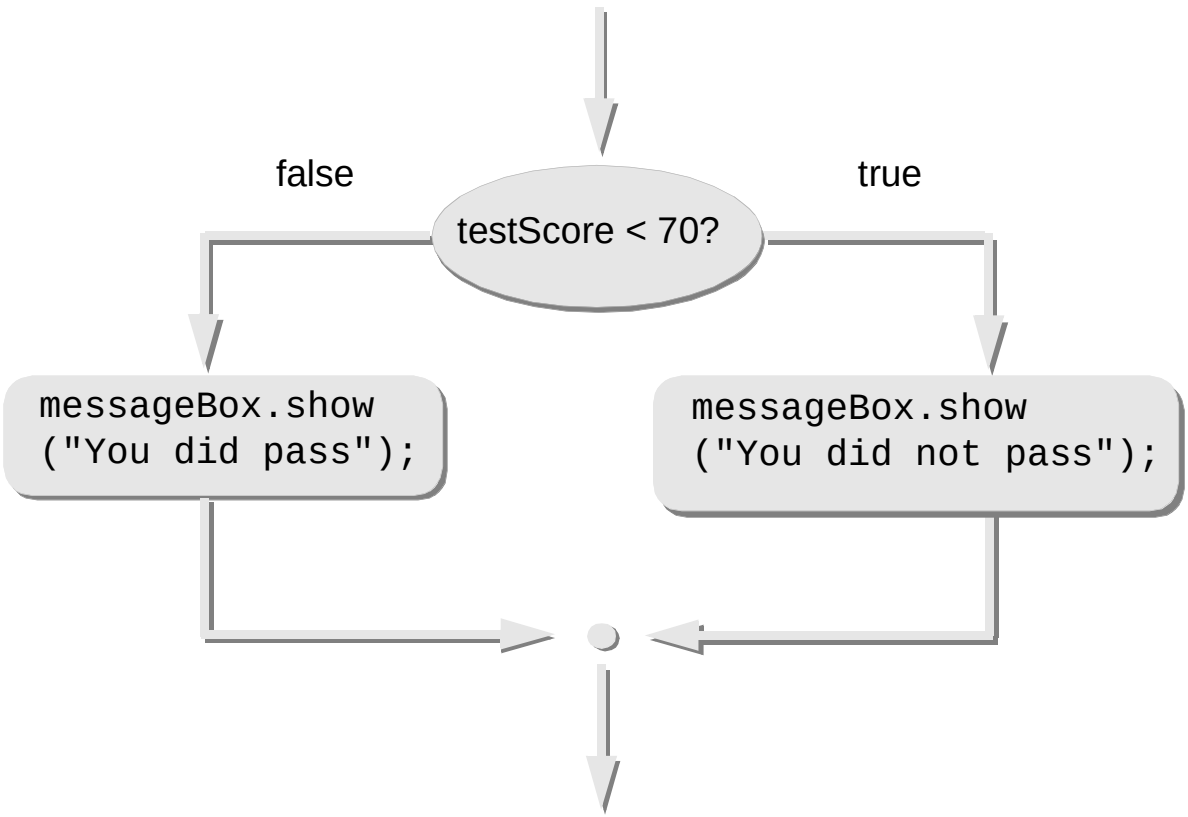


FIGURE 6.3 The diagram showing the control flow of the second version of the **if** statement.

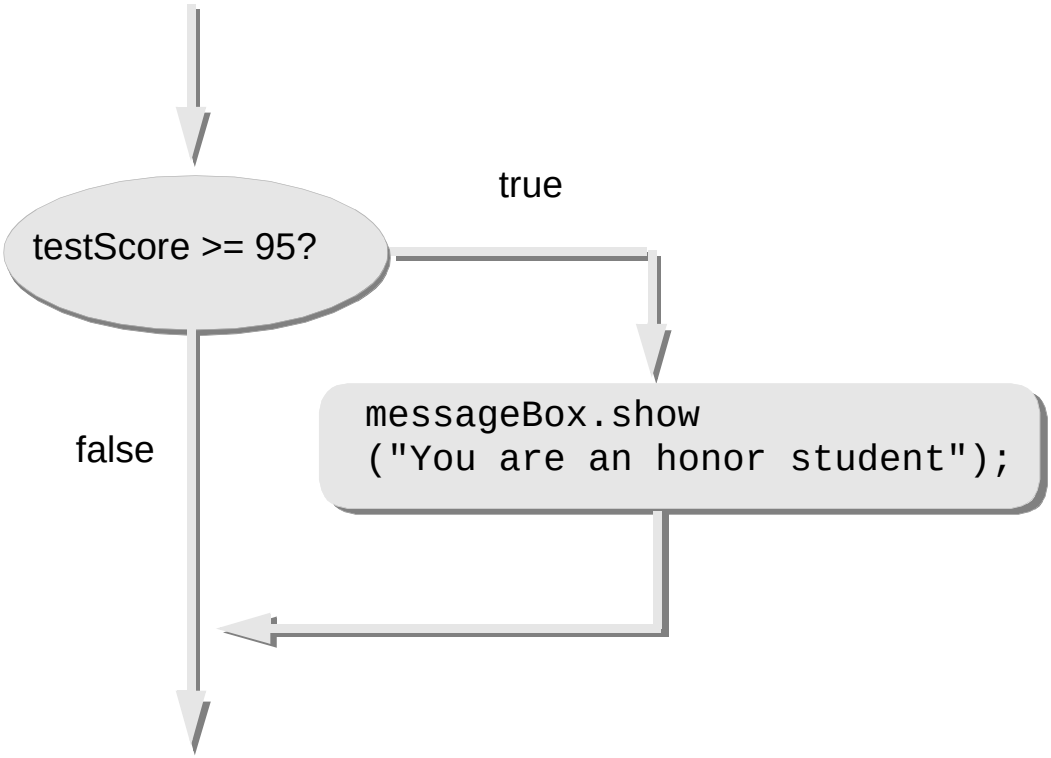


TABLE 6.1 Boolean operators and their meanings.

A	B	A && B	A || B	!A
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

TABLE 6.2 Operator precedence rules. Groups are listed in descending order of precedence. An operator with a higher precedence will be evaluated first. If two operators have the same precedence, then their associativity rule is applied.

Group	Operator	Precedence	Associativity
subexpression	( )	9 (If parentheses are nested, then innermost subexpression is evaluated first.)	left to right
unary operators	- !	8	right to left
multiplicative operators	* / %	7	left to right
additive operators	+ -	6	left to right
comparison operators	< <= > >=	5	left to right
equality operators	== !=	4	left to right
boolean AND	&&	3	left to right
boolean OR	||	2	left to right
assignment	=	1	right to left

FIGURE 6.4 A diagram showing the control flow of the example nested-if statement.

```
if (testScore >= 70) {
   if (studentAge < 10) {
      messageBox.show("You did a great job");
   }
   else {
      messageBox.show("You did pass"); //test score >= 70
   }                                    //and age >= 10
}
else { //test score < 70

   messageBox.show("You did not pass");
}
```

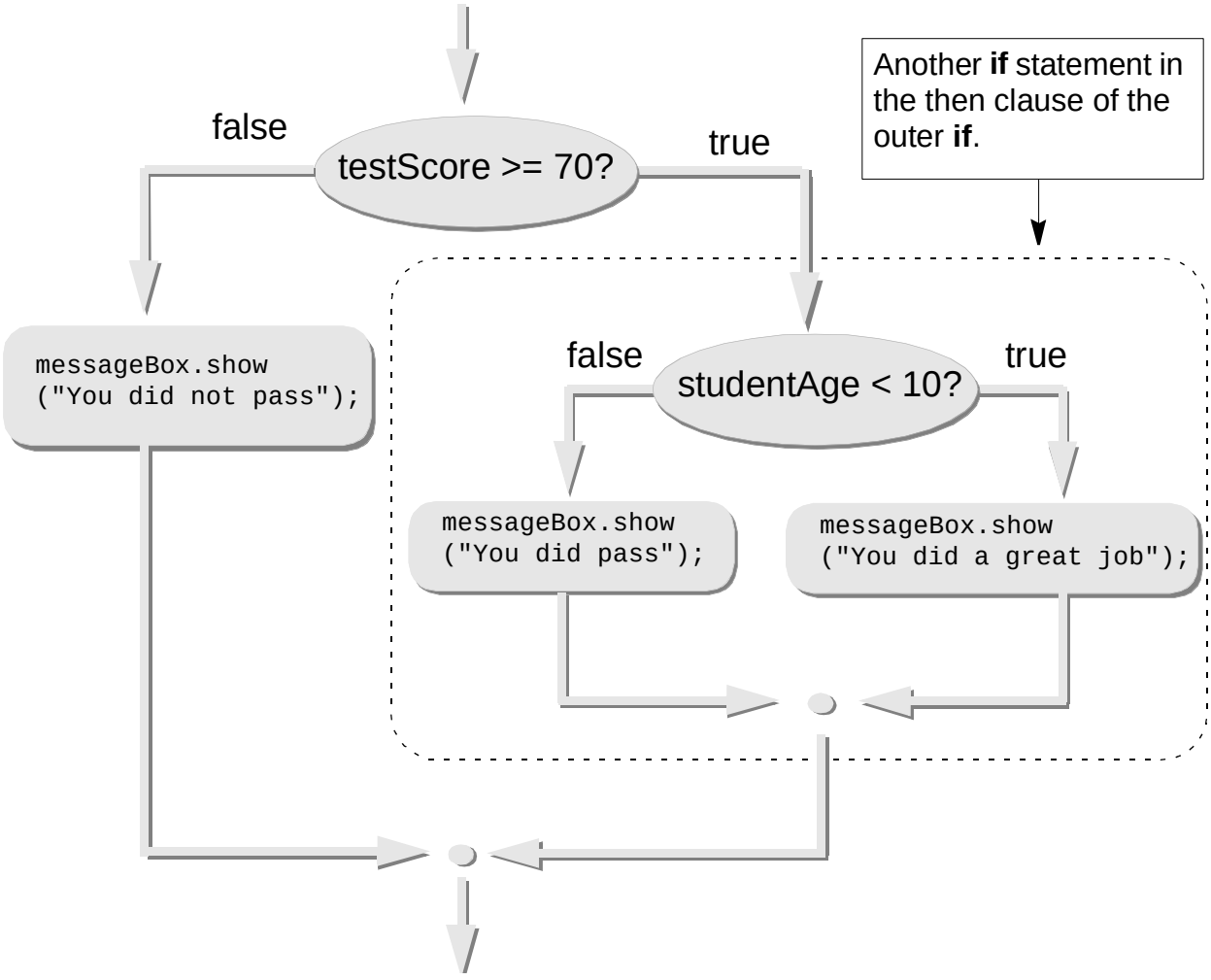
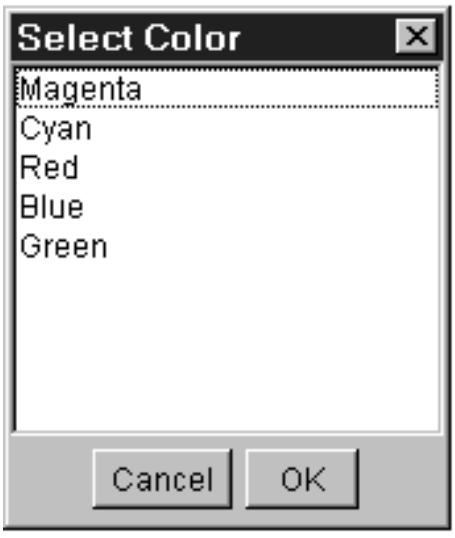


FIGURE 6.5 The **ListBox** object **colorList**. The figure on the right shows the index values of the items in the list.

```
colorList.addItem("Magenta");
colorList.addItem("Cyan");
colorList.addItem("Red");
colorList.addItem("Blue");
colorList.addItem("Green");
selection =
colorList.getSelectedIndex();
```



Index value

0	Magenta
1	Cyan
2	Red
3	Blue
4	Green

TABLE 6.3 A partial list of **ListBox** methods.

CLASS: ListBox		
Method	**Argument**	**Description**
`<constructor>`	`MainWindow`	Creates a ListBox object.
`addItem`	`String`	Adds the argument String value to the list. Items added to the list from top to bottom. The topmost item has the index value of zero, the next item's value is one, and so forth.
`getSelectedIndex`	`<none>`	Returns the index value of the selected item in the list. See the explanation of class constants.

Class Constant	**Description**
`NO_SELECTION`	This value is returned by getSelectedIndex method when the user clicks the OK button without selecting a choice.
`CANCEL`	This value is returned by the getSelectedIndex method when the user clicks the CANCEL button or the dialog's close box.

FIGURE 6.6 Mapping of the sample **switch** statement to the general format.

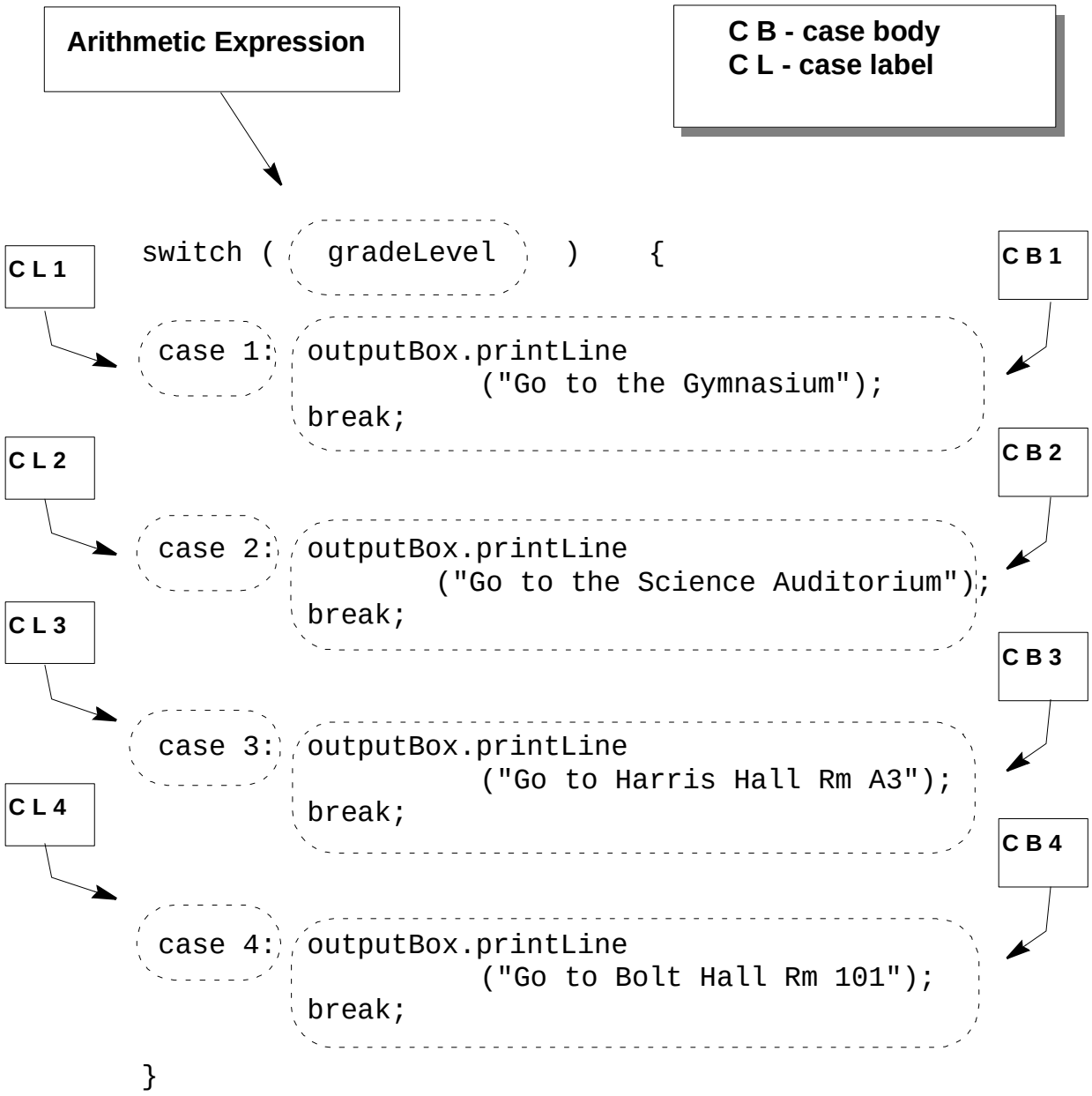
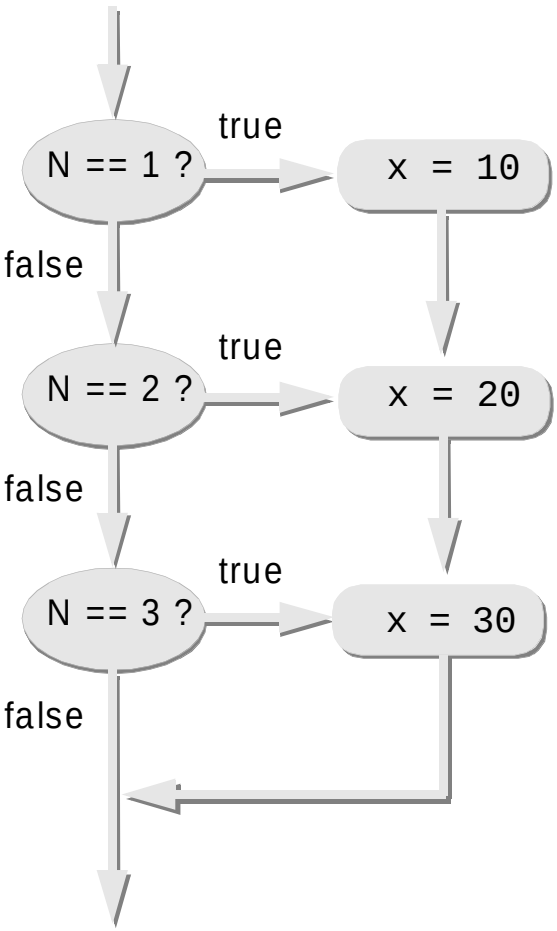


FIGURE 6.7 A diagram showing the control flow of the **switch** statement with and without the break statements.

```
switch ( N ) {
 case 1: x = 10;
 case 2: x = 20;
 case 3: x = 30;
}
```




```
switch ( N ) {
 case 1: x = 10; break;
 case 2: x = 20; break;
 case 3: x = 30; break;
}
```

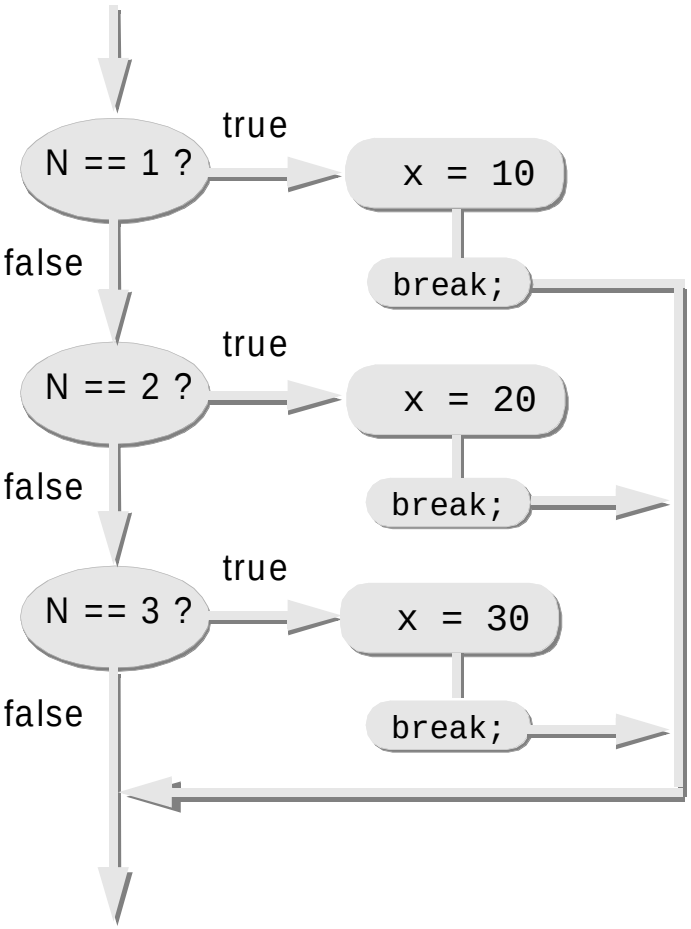


FIGURE 6.8 The object diagram for the **DrawShape** program.