

4.8 while Iteration Statement

An iteration statement allows you to specify that a program should repeat an action while some condition remains *true*. The pseudocode statement

```
While there are more items on my shopping list  
Purchase next item and cross it off my list
```

describes the iteration during a shopping trip. The condition “there are more items on my shopping list” may be *true* or *false*. If it’s *true*, then the action “Purchase next item and cross it off my list” is performed. This action will be performed *repeatedly* while the condition remains *true*. The statement(s) contained in the *While* iteration statement constitute its body, which may be a single statement or a block. Eventually, the condition will become *false* (when the shopping list’s last item has been purchased and crossed off). At this point, the iteration terminates, and the first statement after the iteration statement executes.

As an example of Java’s **while iteration statement**, consider a program segment that finds the first power of 3 larger than 100. After the following **while** statement executes, **product** contains the result:

```
int product = 3;

while (product <= 100) {
    product = 3 * product;
}
```

Each iteration of the `while` statement multiplies `product` by 3, so `product` takes on the values 9, 27, 81 and 243 successively. When `product` becomes 243, `product <= 100` becomes false. This terminates the iteration, so the final value of `product` is 243. At this point, program execution continues with the next statement after the `while` statement.



Common Programming Error 4.2

*Not providing in the body of a `while` statement an action that eventually causes the condition in the `while` to become false normally results in a logic error called an **infinite loop** (the loop never terminates).*

UML Activity Diagram for a `while` Statement

The UML activity diagram in [Fig. 4.6](#) illustrates the flow of

control in the preceding `while` statement. Once again, the symbols in the diagram (besides the initial state, transition arrows, a final state and three notes) represent an action state and a decision. This diagram introduces the UML's **merge symbol**. The UML represents both the merge symbol and the decision symbol as diamonds. The merge symbol joins two flows of activity into one. In this diagram, the merge symbol joins the transitions from the initial state and from the action state, so they both flow into the decision that determines whether the loop should begin (or continue) executing.

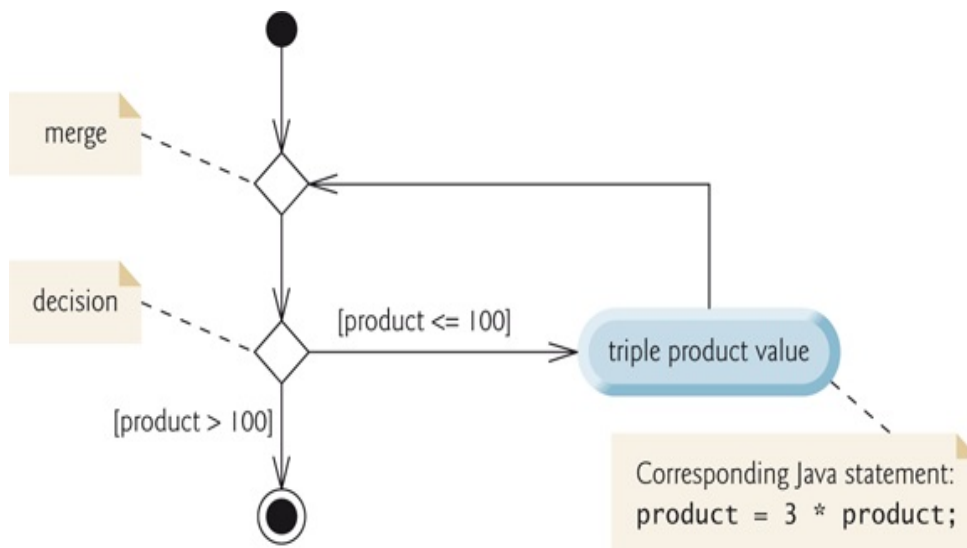


Fig. 4.6

`while` iteration statement UML activity diagram.

Description

The decision and merge symbols can be distinguished by the number of “incoming” and “outgoing” transition arrows. A

decision symbol has one transition arrow pointing to the diamond and two or more pointing out from it to indicate possible transitions from that point. In addition, each transition arrow pointing out of a decision symbol has a guard condition next to it. A merge symbol has two or more transition arrows pointing to the diamond and only one pointing from the diamond, to indicate multiple activity flows merging to continue the activity. *None* of the transition arrows associated with a merge symbol has a guard condition.

Figure 4.6 clearly shows the iteration of the `while` statement discussed earlier in this section. The transition arrow emerging from the action state points back to the merge, from which program flow transitions back to the decision that's tested at the beginning of each iteration of the loop. The loop continues to execute until the guard condition `product > 100` becomes true. Then the `while` statement exits (reaches its final state), and control passes to the next statement in sequence in the program.