# 5.10 Structured-Programming Summary

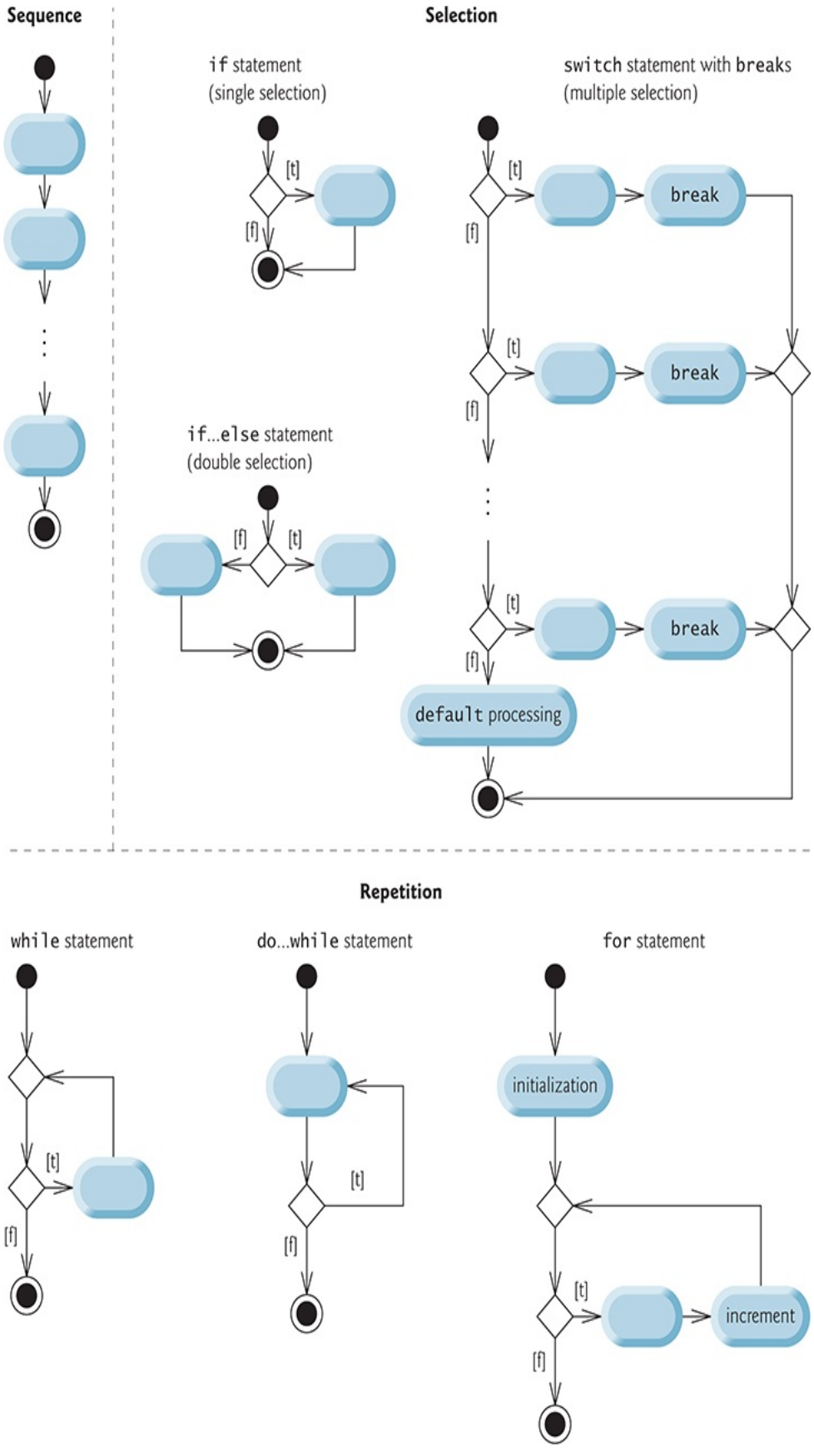
Just as architects design buildings by employing the collective wisdom of their profession, so should programmers design programs. Our field is much younger than architecture, and our collective wisdom is considerably sparser. We've learned that structured programming produces programs that are easier than unstructured programs to understand, test, debug, modify and even prove correct in a mathematical sense.

# Java Control Statements Are Single-Entry/Single-Exit

Figure 5.21 uses UML activity diagrams to summarize Java's control statements. The initial and final states indicate the *single entry point* and the *single exit point* of each control statement. Arbitrarily connecting individual symbols in an activity diagram can lead to unstructured programs. Therefore, the programming profession has chosen a limited set of control statements that can be combined in only two simple ways to build structured programs.

For simplicity, Java includes only *single-entry/single-exit* control statements—there's only one way to enter and only

one way to exit each control statement. Connecting control statements in sequence to form structured programs is simple. The final state of one control statement is connected to the initial state of the next—that is, the control statements are placed one after another in a program in sequence. We call this *control-statement stacking*. The rules for forming structured programs also allow for control statements to be *nested*.


Sequence
Selection
if statement
(single selection)
[t]
[f]
switch statement with breaks
(multiple selection)
break
break
break
default processing
if...else statement
(double selection)
[f]
[t]
Repetition
while statement
do...while statement
for statement
initialization
increment

# Fig. 5.21

Java’s single-entry/single-exit sequence, selection and iteration statements.

Description

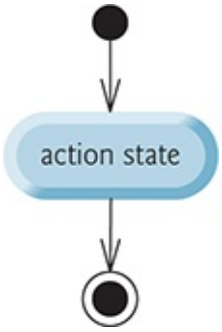
# Rules for Forming Structured Programs

Figure 5.22 shows the rules for forming structured programs. The rules assume that action states may be used to indicate *any* action. The rules also assume that we begin with the simplest activity diagram (Fig. 5.23) consisting of only an initial state, an action state, a final state and transition arrows.

| Rules for forming structured programs | |
|---|---|
| 1. | Begin with the simplest activity diagram (Fig. 5.23). |
| 2. | Any action state can be replaced by two action states in sequence. |
| 3. | Any action state can be replaced by any control statement (`if`, `if`…`else`, `switch`, `while`, `do`…`while` or `for`). |
| 4. | Rules 2 and 3 can be applied as often as you like and in any order. |

# Fig. 5.22

Rules for forming structured programs.




# Fig. 5.23

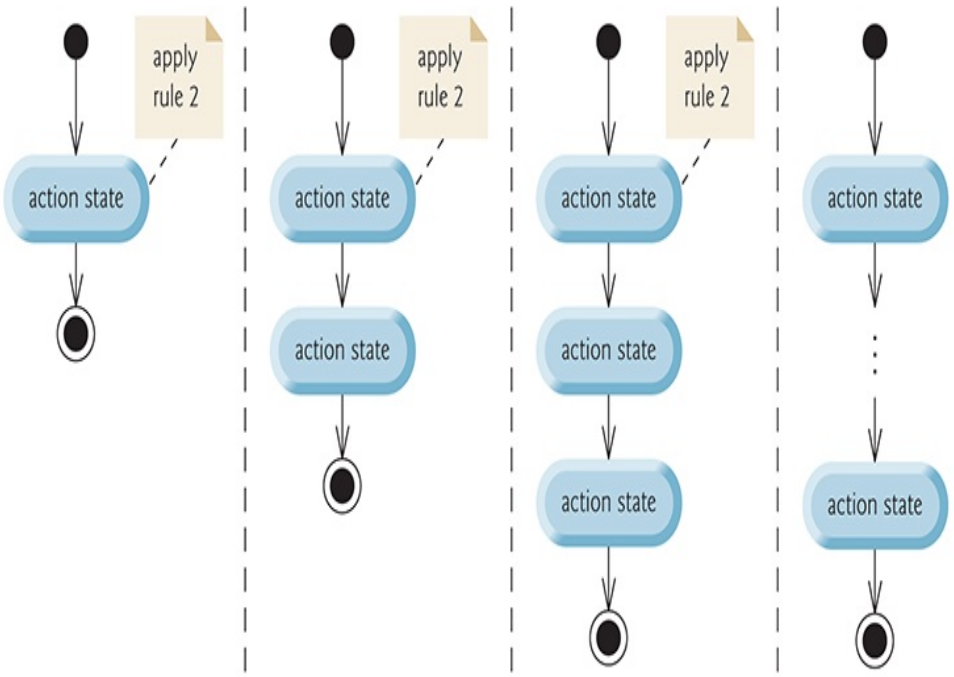
Simplest activity diagram.

Applying the rules in Fig. 5.22 always results in a properly structured activity diagram with a neat, building-block appearance. For example, repeatedly applying rule 2 to the simplest activity diagram results in an activity diagram containing many action states in sequence (Fig. 5.24). Rule 2 generates a *stack* of control statements, so let's call rule 2 the **stacking rule**. The vertical dashed lines in Fig. 5.24 are not part of the UML—we use them to separate the four activity diagrams that demonstrate rule 2 of Fig. 5.22 being applied.

Rule 3 is called the **nesting rule**. Repeatedly applying rule 3 to the simplest activity diagram results in one with neatly *nested* control statements. For example, in Fig. 5.25, the action state in the simplest activity diagram is replaced with a

double-selection (`if...else`) statement. Then rule 3 is applied again to the action states in the double-selection statement, replacing each with a double-selection statement. The dashed action-state symbol around each double-selection statement represents the action state that was replaced. [*Note:* The dashed arrows and dashed action-state symbols shown in Fig. 5.25 are not part of the UML. They're used here to illustrate that *any* action state can be replaced with a control statement.]

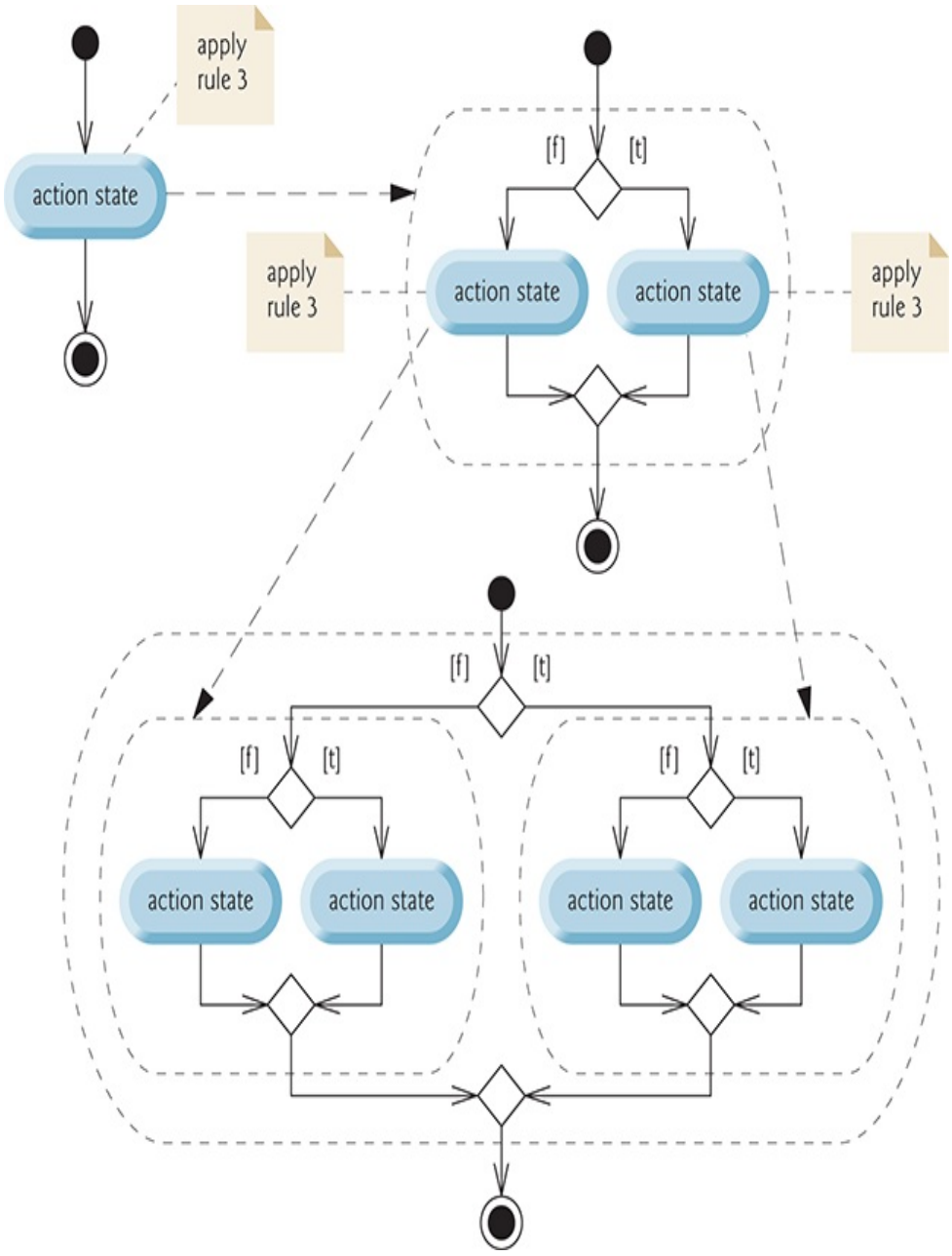
Rule 4 generates larger, more involved and more deeply nested statements. The diagrams that emerge from applying the rules in Fig. 5.22 constitute the set of all possible structured activity diagrams and hence the set of all possible structured programs. The beauty of the structured approach is that we use *only seven* simple single-entry/single-exit control statements and assemble them in *only two* simple ways.



# Fig. 5.24

Repeatedly applying rule 2 of Fig. 5.22 to the simplest activity diagram.

Description

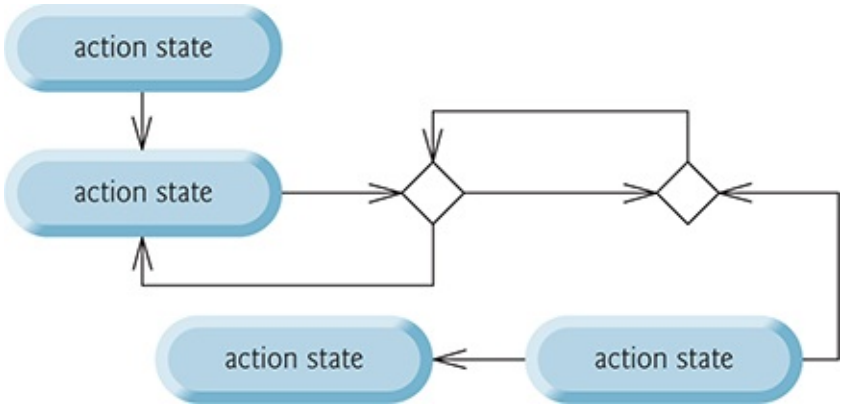


# Fig. 5.25

Repeatedly applying rule 3 of Fig. 5.22 to the simplest activity diagram.

Description

If the rules in Fig. 5.22 are followed, an “unstructured’ activity diagram (like the one in Fig. 5.26) cannot be created. If you’re uncertain about whether a particular diagram is structured, apply the rules of Fig. 5.22 in reverse to reduce it to the simplest activity diagram. If you can reduce it, the original diagram is structured; otherwise, it’s not.




# Fig. 5.26

“Unstructured” activity diagram.

Description

# Three Forms of Control

Structured programming promotes simplicity. Only three forms of control are needed to implement an algorithm:

- sequence
- selection
- iteration

The sequence structure is trivial. Simply list the statements to execute in the order in which they should execute. Selection is implemented in one of three ways:

- `if` statement (single selection)
- `if`…`else` statement (double selection)
- `switch` statement (multiple selection)

In fact, it's straightforward to prove that the simple `if` statement is sufficient to provide *any* form of selection—everything that can be done with the `if`…`else` statement and the `switch` statement can be implemented by combining `if` statements (although perhaps not as clearly and efficiently).

Iteration is implemented in one of three ways:

- `while` statement
- `do`…`while` statement
- `for` statement

[*Note:* There's a fourth iteration statement—the *enhanced* `for`

*statement*—that we discuss in Section 7.7.] It's straightforward to prove that the `while` statement is sufficient to provide *any* form of iteration. Everything that can be done with `do`…`while` and `for` can be done with the `while` statement (although perhaps not as conveniently).

Combining these results illustrates that *any* form of control ever needed in a Java program can be expressed in terms of

- sequence
- `if` statement (selection)
- `while` statement (iteration)

and that these can be combined in only two ways—*stacking* and *nesting*. Indeed, structured programming is the essence of simplicity.