Chapter3

Tuesday, October 7, 2025 3:35 PM

Switch

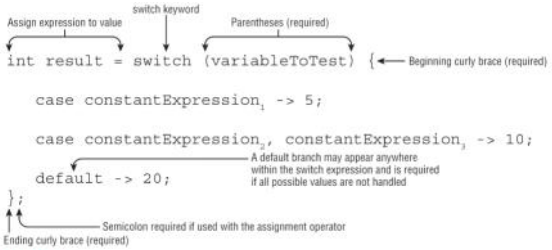


FIGURE 3.5 A `switch` expression

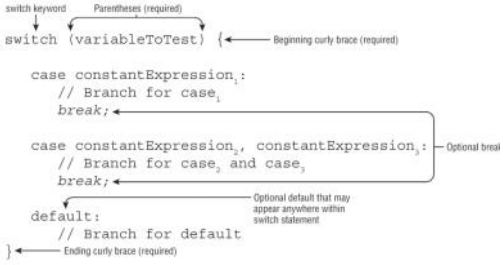


FIGURE 3.4 A `switch` statement

```
final int getCookies() { return 4; }
void feedAnimals() {
   final int bananas = 1;
   int apples = 2;
   int numberOfAnimals = 3;
   final int cookies = getCookies();
   switch (numberOfAnimals) {
      case bananas:
      case apples:       // DOES NOT COMPILE
      case getCookies(): // DOES NOT COMPILE
      case cookies :     // DOES NOT COMPILE
      case 3 * 5 :
   } }
```

When writing `switch` expressions, it may be a good idea to add a `default` branch, even if you cover all possible values. This means that if someone modifies the enum with a new value, your code will still compile.

```
Number variableToTest = ...

int result = switch (variableToTest) {
   case Integer x when x > 0 && x <=20 -> 3;

   case Integer x -> 5;

   case Number n -> 10;
};
```

Type matches switch variable reference type so default clause is not needed

FIGURE 3.7 Pattern matching with `switch`

Properties:

1. Must to be @Exaustive.
2. Yield is used for return a value from a case block expression. If we will use return, then he whole method will return the value.

WATCH SEMICOLONS IN *SWITCH* EXPRESSIONS

When writing a `case` expression, a semicolon is required, but when writing a `case` block, it is prohibited.

```
int fish = 1;
var name = switch (fish) {
   case 1  -> "Goldfish"          // DOES NOT COMPILE (missing s
   case 2  -> { yield "Trout"; };  // DOES NOT COMPILE (extra sem
   default -> "Shark";
}  // DOES NOT COMPILE (missing semicolon)
```

A bit confusing, right? It's just one of those things you have to train yourself to spot on the exam.

3. Unreachable code will throw an compile time error.
4. Have a null case in Java21. To avoid boilerplate code.

New to Java 21, `switch` now supports `case null` clause when working with object types, allowing us to rewrite our previous example as the following:

```
String fish = null;
System.out.print(switch (fish) {
   case "ClownFish" -> "Hello!";
   case "BlueTang"  -> "Hello again!";
   case null        -> "What type of fish are you?";
   default          -> "Goodbye";
});
```

That's a lot less boilerplate code, now that we don't have to handle `null` separately.

5. @Case null is allowed just in pattern matching switch, and the order in case null matter. Needs to be the last and after a default case to ensure will cover all the posibilities.

```
int result = switch (variableToTest) {
   case constantExpression1 -> 5;

   case constantExpression2, constantExpression3 -> {
      yield 10;
   }
   ...
};
```

case expression — case block — A yield is required within a case block unless an exception is thrown — Curly braces required for case blocks

FIGURE 3.6 A `switch` expression with a `case` block and `yield` statement

WATCH SEMICOLONS IN *SWITCH* EXPRESSIONS

When writing a `case` expression, a semicolon is required, but when writing a `case` block, it is prohibited.

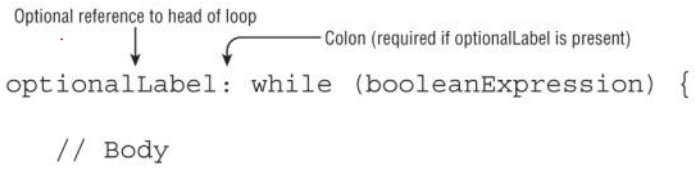
```
int fish = 1;
var name = switch (fish) {
   case 1  -> "Goldfish"          // DOES NOT COMPILE (missing s
   case 2  -> { yield "Trout"; };  // DOES NOT COMPILE (extra sem
   default -> "Shark";
}  // DOES NOT COMPILE (missing semicolon)
```

A bit confusing, right? It's just one of those things you have to train yourself to spot on the exam.

FOR Loops:

4. Using Incompatible Data Types in the Initialization Block

```
int x = 0;
for (long y = 0, int z = 4; x < 5; x++)  // DOES NOT COMPILE
   System.out.print(y + " ");
```



Summary

Concept	What it affects	Can you change the reference?	Can you change the object?
`String`	object immutability	✅ yes	❌ no
`final String`	reference finality +	❌ no	❌ no

```
// Body

// Somewhere in the loop
break optionalLabel;
```

break keyword — Semicolon (required)

}

FIGURE 3.12 The structure of a `break` statement

`String`	object immutability	✅ yes	❌ no
`final String`	reference finality + object immutability	❌ no	❌ no
`final StringBuilder`	reference finality only	❌ no	✅ yes

EXAM

1.F.
2.A,B,C
3.B.
4.A,D,F
5.C. -> F
6.C. -> E
7.B,D
8.G.
9.E,A
10.E
11.D (Cred ca return typul trebuie sa fie exact cu cel asteptat.
12.C
13.F
14.G
15.B,D,F
16.F.
17.d,B,A
18.e,b
19.F

19. E. The variable `snake` is declared within the body of the `do/while` statement, so it is out of scope on line 7. For this reason, option E is the correct answer. If `snake` were declared before line 3 with a value of `1`, then the output would have been `1 2 3 4 5 -5.0`, and option G would have been the correct answer.

```
2: double iguana = 0;
3: do {
4:    int snake = 1;
5:    System.out.print(snake++ + " ");
6:    iguana--;
7: } while (snake <= 5);
8: System.out.println(iguana);
```

A. `1 2 3 4 -4.0`
B. `1 2 3 4 -5.0`
C. `1 2 3 4 5 -4.0`
D. `0 1 2 3 4 5 -5.0`
E. The code does not compile.
F. The code compiles but produces an infinite loop at runtime.
G. None of the above.

20. A,E

```
4:  int height = 1;
5:  L1: while (height++ <10) {
6:     long humidity = 12;
7:     L2: do {
8:        if (humidity-- % 12 == 0) ;
9:        int temperature = 30;
10:       L3: for ( ; ; ) {
11:          temperature++;
12:          if (temperature>50) ;
13:       }
14:    } while (humidity> 4);
15: }
```

A. `break L2` on line 8; `continue L2` on line 12
B. `continue` on line 8; `continue` on line 12
C. `break L3` on line 8; `break L1` on line 12
D. `continue L2` on line 8; `continue L3` on line 12
E. `continue L2` on line 8; `continue L2` on line 12
F. None of the above, as the code contains a compiler error

Option C is incorrect because it contains a compiler error. The label `L3` is not visible outside its loop. Option D is incor-

21.D

21. A minimum of how many lines need to be corrected before the following

21. A minimum of how many lines need to be corrected before the following method will compile?

```
21: void findZookeeper(Integer id) {
22:    System.out.print(switch (id) {
23:       case 10 -> {"Jane";}
24:       case 20 -> {yield "Lisa";};
25:       case 30 -> "Kelly";
26:       case 30 -> "Sarah";
27:       default -> "Unassigned";
28:    });
29: }
```

A. Zero
B. One
C. Two
D. Three
E. Four
F. Five

21. D. Line 23 does not compile because it is missing a `yield` statement. Line 24 does not compile because it contains an extra semicolon at the end. Finally, lines 25 and 26 do not compile because they use the same `case` value. At least one of them would need to be changed for the code to compile. Since three lines need to be corrected, option D is correct.

22.G E

22. What is the output of the following code snippet?

```
2: var tailFeathers = 3;
3: final var one = 1;
4: switch (tailFeathers) {
5:    case one: System.out.print(3 + " ");
6:    default: case 3: System.out.print(5 + " ");
7: }
8: while (tailFeathers> 1) {
9:    System.out.print(--tailFeathers + " "); }
```

A. `3`
B. `5 1`
C. `5 2`
D. `3 5 1`
E. `5 2 1`
F. The code will not compile because of lines 3–5.
G. The code will not compile because of line 6.

23.D F

23. What is the output of the following code snippet?

```
15: int penguin = 50, turtle = 75;
16: boolean older = penguin>= turtle;
17: if (older = true) System.out.println("Success");
18: else System.out.println("Failure");
19: else if (penguin != 50) System.out.println("Other");
```

A. `Success`
B. `Failure`
C. `Other`
D. The code will not compile because of line 17.
E. The code compiles but throws an exception at runtime.
F. None of the above.

24.B

24. What is the output of the following code snippet?

```
22: String zooStatus = "Closed";
23: int visitors = switch (zooStatus) {
24:    case String s when s.equals("Open") -> 10;
25:    case Object s when s != null && !s.equals("") -> 20;
26:    case null -> {yield 30;}
27:    default -> 40;
28: };
29: System.out.print(visitors);
```

A. `10`
B. `20`
C. `30`
D. `40`
E. Exactly one line does not compile.
F. Exactly two lines do not compile.

24. B. Since this is a pattern matching `switch` statement, the `case` branches are evaluated in the order in which they appear. In particular, each branch does not dominate the ones after it, so the code compiles without issue. If either of the `when` clauses were removed from their accompanying `case` clause, then the code would not compile. The first branch is skipped because `Closed` does not match `Open`. The second one matches, resulting in `20` being printed at runtime and making option B correct.

25.D

25. What is the output of the following code snippet?

```
6:  String instrument = "violin";
7:  final String CELLO = "cello";
8:  String viola = "viola";
9:  int p = -1;
10: switch (instrument) {
11:    case "bass" : break;
12:    case CELLO : p++;
13:    default: p++;
14:    case "VIOLIN": p++;
15:    case "viola" : ++p; break;
16: }
17: System.out.print(p);
```

marked `final`. Since `"violin"` and `"VIOLIN"` are not an exact match, the `default` branch of the `switch` statement is executed at runtime. This execution path increments `p` a total of three times, bringing the final value of `p` to `2` and making option D the correct answer.

!!!Attention
If no break, then all below cases will be executed, indifferently of the condition case.

26.F

26. What is the output of the following code snippet?

```
9:  int w = 0, r = 1;
10: String name = "";
11: while (w < 2) {
12:    name += "A";
13:    do {
14:       name += "B";
15:       if (name.length()>0) name += "C";
16:       else break;
17:    } while (r <=1);
18:    r++; w++; }
19: System.out.println(name);
```

A. `ABC`
B. `ABCABC`
C. `ABCABCABC`
D. Line 15 contains a compilation error.
E. Line 18 contains a compilation error.
F. The code compiles but never terminates at runtime.
G. The code compiles but throws a `NullPointerException` at runtime.

27.D

28.F

28. What is the output of calling `getFish("goldie")`?

```
40: void getFish(Object fish) {
41:    if (!(fish instanceof String guppy))
42:       System.out.print("Eat!");
43:    else if (!(fish instanceof String guppy)) {
44:       throw new RuntimeException();
45:    }
46:    System.out.print("Swim!");
47: }
```

A. `Eat!`
B. `Swim!`
C. `Eat!` followed by an exception
D. `Eat!Swim!`
E. An exception is printed
F. None of the above

28. F. Based on flow scoping, `guppy` is in scope after lines 41–42 if the type is not a `String`. In this case, line 43 declares a variable `guppy` that is a duplicate of the previously defined local variable defined on line 41. For this reason, the code does not compile, and option F is correct. If a different variable name was used on line 43, then the code would compile and print `Swim!` at runtime with the specified input.

The problem is the guppy appears 2 times with the same name.

```java
if (!(fish instanceof String guppy)) {
    // guppy is NOT in scope ❌
} else {
    // guppy IS in scope ✅
}
```

```
if (fish instanceof String guppy) {
    // guppy is in scope here ✅
} else {
    // guppy is NOT in scope here ❌
}
```

29. C

29. What is the result of the following code?

```
1: public class PrintIntegers {
2:    public static void main(String[] args) {
3:       int y = -2;
4:       do System.out.print(++y + " ");
5:       while (y <= 5);
6: } }
```

This will compile even we don't have " {}"

A. `-2 -1 0 1 2 3 4 5`
B. `-2 -1 0 1 2 3 4`
C. `-1 0 1 2 3 4 5 6`
D. `-1 0 1 2 3 4 5`
E. The code will not compile because of line 5.
F. The code contains an infinite loop and does not terminate.

30.

30. What is the minimum number of lines that would need to be changed or removed for the following code to compile and return a value when called with `dance(10)`?

```
41: double dance(Object speed) {
42:    return switch (speed) {
43:       case 5 -> {yield 4};
44:       case 10 -> 8;
45:       case 15,20 -> 12;
46:       default -> 20;
47:       case null -> 16;
48:    }
49: }
```

A. Zero, the code compiles and runs without issue
B. One
C. Two
D. Three
E. Four
F. Five
G. Six

30. E. On line 43, the semicolon should be after the `yield` statement, not outside the brace. Line 48 is missing a semicolon after the `return` statement containing the `switch` expression. For these reasons, at least two lines must be corrected. Next, lines 43, 44, and 45 do not compile because the numeric values are not compatible with the reference type for `Object`. We can fix this by changing line 41 to pass `speed` as a compatible type, such as `Integer`. Finally, the `default` clause on line 46 dominates the proceeding `case null` on line 47. Removing line 47 fixes this issue, as `case null` is not required. Since we can get the code to compile by changing or removing four lines, option E is the correct answer.