

Statements

Purposes of statements:

Declaring variables and assigning values.

Control **whether** code will be executed.

Control **how often** code will be executed.

Statements: General syntax

Statement's body terminated by “;”

```
{statement};
```

Statement examples: Declaring and assigning variables

Variable declaration:

```
int a;
```

Value assignment:

```
a = 33;
```

Combined declaration and assignment:

```
int a = 33;
```

Expression vs. statement

Expression:

```
a - 4
```

Statement:

```
b = a - 4;
```

Notice the trailing “;”.

Multiple statements per line

```
a = b + 3; b = a - 4;
```

Discouraged by good coding practices:

- Poor readability
- Hampers debugging

Debugging multiple statements per line

The screenshot shows an IDE window for a project named "Dummy" in the "src" directory, with the "Main" package selected. The file "Main.java" is open, and the code is as follows:

```
2 public static void main(String[] args) { args: {} ✓
3     int a = 1, b; a: 1
4     a *= 3; b = ++a - 4; a: 1
5     System.out.println(b);
```

The debugger is currently paused at line 4. The "Debug" toolbar shows the "Debugger" tab selected, with the "Console" tab also visible. The "Frames" pane shows the current frame as "main:4, Main". The "Variables" pane shows the current state of the program:

- `args = {String[0]@431}`
- `a = 1`

Class scope

- Variable being defined on class level.
- Visible to all methods.

```
public class X {  
    // Class variable  
    static int i = 3;  
  
    public static void main(String[] args) {  
        System.out.println("main: i=" + i);  
        someMethod();  
    }  
    public static void someMethod() {  
        System.out.println("someMethod: i=" + i);  
    }  
}
```

Method local variable scope

- Method scope being delimited by the { ... } block
- A variable's visibility is restricted to its block:

```
public class X {  
  
    public static void main(String[] args) {  
        int i = 1; // Visible in current method  
        System.out.println("i=" + i);  
        someMethod(); // Method call  
    }  
  
    public static void someMethod() {  
        int j = 3;  
        System.out.println("j=" + j);  
  
        int i = 17; // No conflict with i in main(...)  
        System.out.println("i=" + i);  
    }  
}
```


Nested blocks

```
public static void main(String[] args) {  
    double initialAmount = 34;  
  
    { // first block  
        final double interestRate = 1.2; // 1.2%  
        System.out.println("Interest:" + initialAmount * interestRate / 100);  
    }  
    { // second block  
        final double interestRate = 0.8; // 0.8%  
        System.out.println("Interest:" + initialAmount * interestRate / 100);  
    }  
}
```

- Defining scopes
- Unit of work

- `if`: Conditional block execution.
- `for / while`: Repeated block execution.

55. Blocks and variable glitches

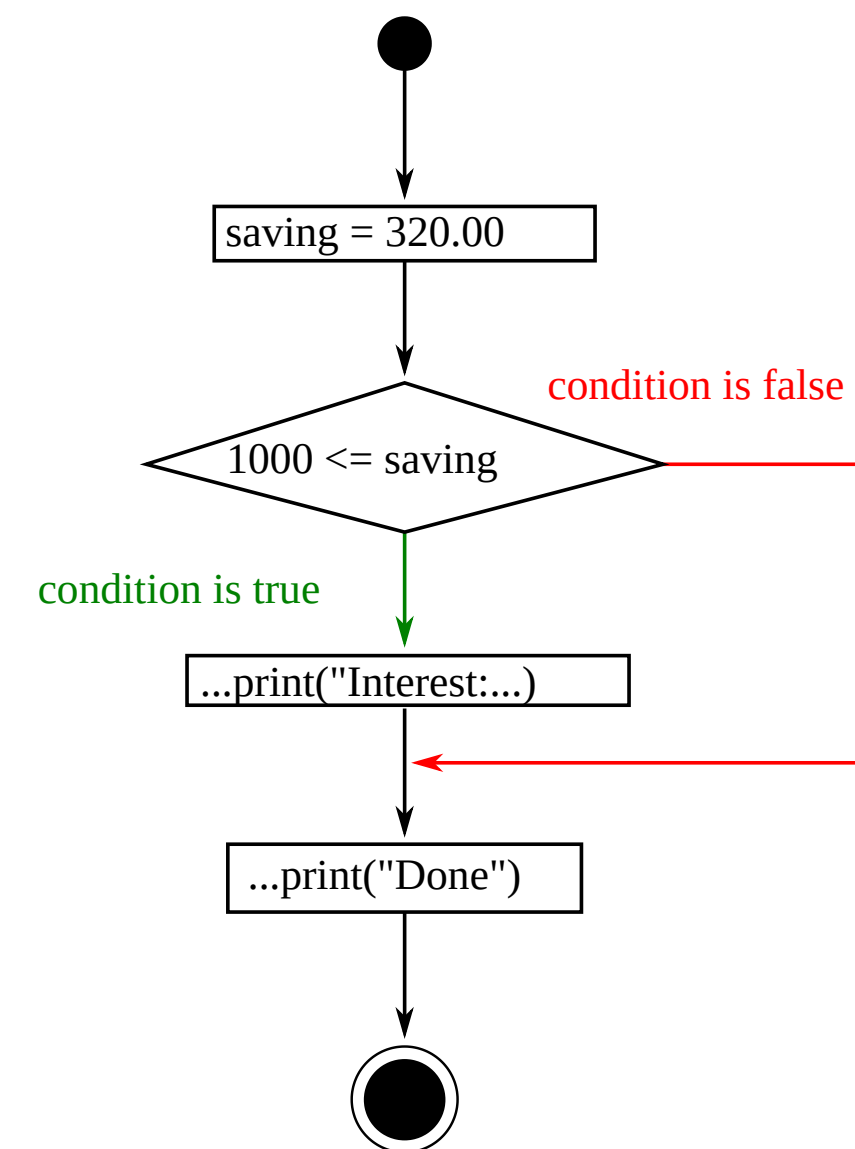
Statements

⇒ The if conditional statement

Conditional block execution

```
double saving = 320.00;  
  
if (1000 <= saving) {  
    // Rich customer, 1,2% interest rate  
    System.out.println(  
        "Interest:" + 1.2 * saving / 100);  
}  
System.out.println("Done!");
```

Done!



if syntax

```
if (booleanExpression)  
  (block | statement)
```

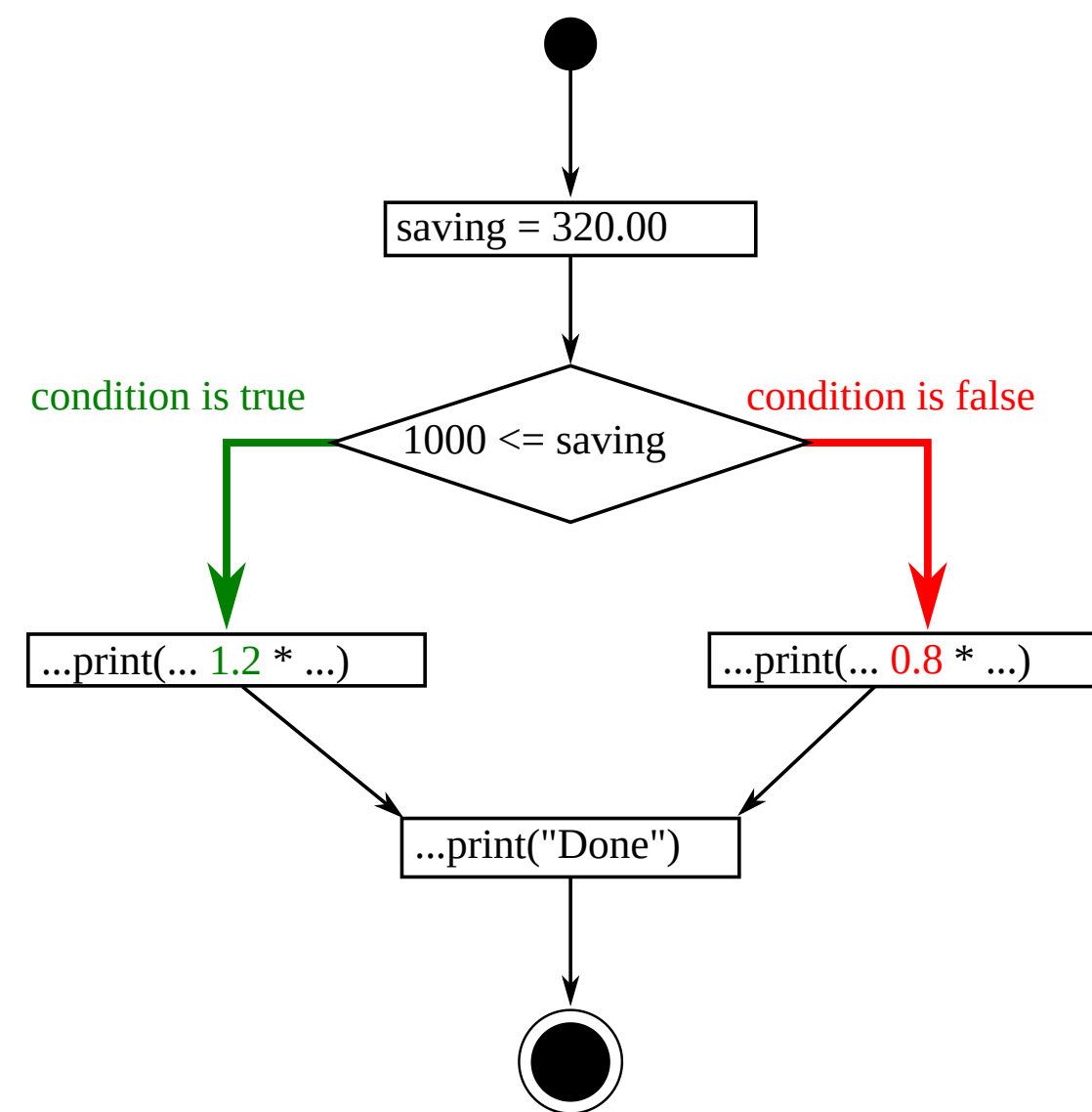
Statements

- ⇒ The if conditional statement
- ⇒ if-then-else

if ... else

```
double saving = 320.00;  
  
if (1000 <= saving ❶) { ❷  
    System.out.println(  
        "Interest:" + 1.2 * saving / 100);  
} ❸ else { ❹  
    System.out.println(  
        "Interest:" + 0.8 * saving / 100);  
}  
System.out.println("Done!");
```

```
Interest:2.56  
Done!
```



if ... else syntax

```
if (booleanExpression)
  (block | statement)
[else
  (block | statement) ] ❶
```


Best practices comparing for equality

Use

```
if (4 == variable) ...
```

in favour of:

```
if (variable == 4) ... 1
```

- 56. Providing better display
- 57. Comparing for equality

Single statement branches

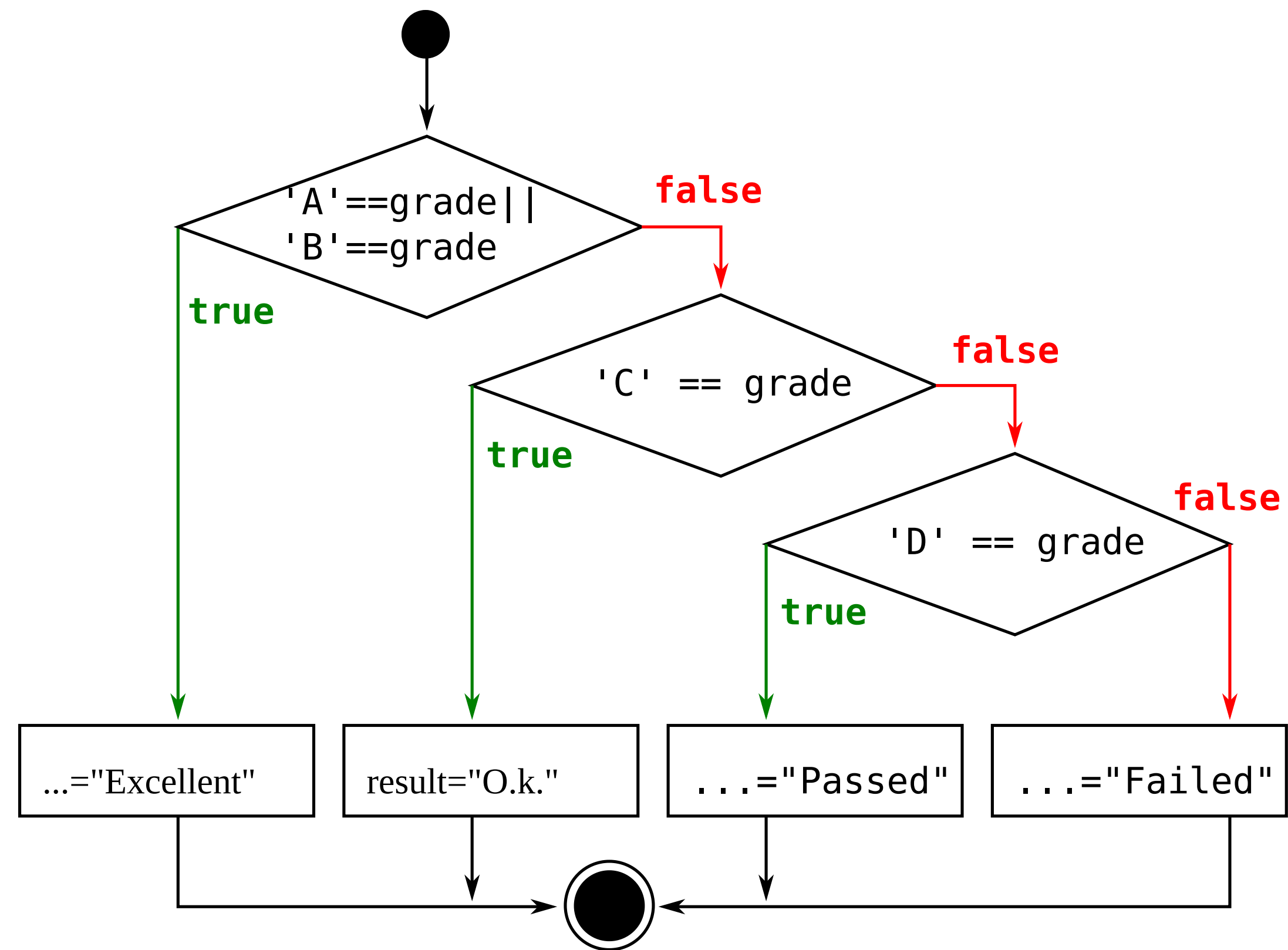
Branches containing exactly one statement don't require a block definition.

```
double initialAmount = 3200;

if (100000 <= initialAmount)
    System.out.println("Interest:" + 1.2 * initialAmount / 100);
else if (1000 <= initialAmount)
    System.out.println("Interest:" + 0.8 * initialAmount / 100);
else
    System.out.println("Interest:" + 0);
```

Nested if ... else

```
if ('A' == grade || 'B' == grade)
  result = "Excellent";
} else {
  if ('C' == grade) {
    result = "O.k.";
  } else {
    if ('D' == grade) {
      result = "Passed";
    } else {
      result = "Failed";
    }
  }
}
```

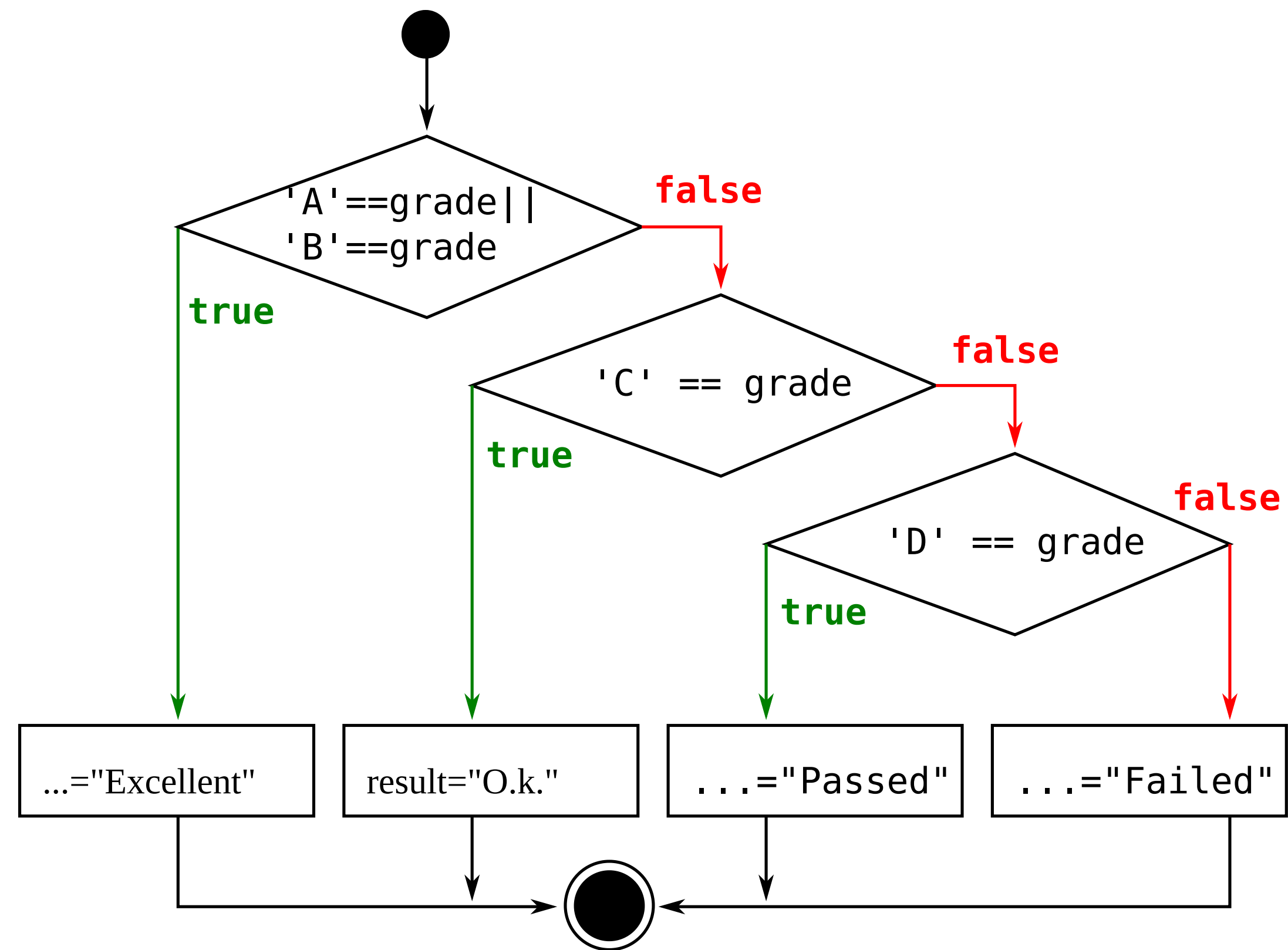


Statements

- ⇒ The if conditional statement
- ⇒ Using `else if`

Enhanced readability: `if ... else if ... else`

```
if ('A' == grade || 'B' == grade)
  result = "Excellent";
} else if ('C' == grade) {
  result = "O.k.";
} else if ('D' == grade) {
  result = "Passed";
} else {
  result = "Failed";
}
```



if ... else if ... else syntax

```
if (booleanExpression)
  (block | statement)
[else if (booleanExpression)
  (block | statement) ]* ❶
[else
  (block | statement) ] ❷
```

58. Replacing `else if (...){...}` by nested `if ... else` statements

User input recipe

```
import java.util.Scanner;

public class App {
    public static void main(String[] args){

        final Scanner scan =
            new Scanner(System.in);
        System.out.print("Enter a value:");
        final int value = scan.nextInt();
        System.out.println("You entered "
            + value);
    }
}
```

```
Enter a value:123
You entered 123
```

See `nextBoolean()`,
`nextByte()` and friends.

Followup exercises

59. Post modifying an exam's marking

60. At the bar

61. Roman numerals

Converting numbers to day's names

Task: Convert day's numbers to day's names

1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday
7	Sunday

Numbers to day's names: The hard way

```
final Scanner scan = new Scanner(System.in);
System.out.print("Enter a weekday number (1=Monday, 2=Tuesday,...) : ");

final int number = scan.nextInt();

if (1 == number) {
    System.out.println("Monday");
} else if (2 == number) {
    System.out.println("Tuesday");
    ...
} else if (7 == number) {
    System.out.println("Sunday");
} else {
    System.out.println("Invalid number " + number);
}
```

62. Leap years

Statements

⇒ The `switch` statement

Better: Using `switch`

```
...
switch(number) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    case 4: System.out.println("Thursday"); break;
    case 5: System.out.println("Friday"); break;
    case 6: System.out.println("Saturday"); break;
    case 7: System.out.println("Sunday"); break;

    default: System.out.println("Invalid number " + number); break;
} ...
```

Enter a weekday number (1=Monday, 2=Tuesday,...) : 6
Saturday

switch Syntax

```
switch(expression) {  
  [case label:  
    [statement]*  
    [break;  
]*  
  [default:  
    [statement]*  
    [break; ]  
}
```


63. Why “break”?

64. Extending to month days

Switching on strings

```
String month, season;
...
// Since Java 7: String based case labels
switch(month) {
    case "March": case "April": case "May":
        season = "Spring"; break;

    case "June": case "July": case "August":
        season = "Summer"; break;

    case "September": case "October": case "November":
        season = "Autumn"; break;

    case "December": case "January": case "February":
        season = "Winter"; break;
}
```

Followup exercises

65. Converting day's names to numbers.
66. Day categories.
67. Roman numerals, using switch

switch expressions

```
switch(number) {  
  case 1: System.out.print("Monday"); break;  
  ...  
  case 7: System.out.print("Sunday"); break;  
  default: System.out.print("Invalid:" + number);  
}
```

```
switch(number) {  
  case 1 -> System.out.print("Monday");  
  ...  
  case 7 -> System.out.print("Sunday");  
  default -> System.out.print("Invalid:" + number);  
}
```

Assigning switch expressions

Code	Output
<pre data-bbox="611 852 1685 1367">int i = 2; // sample value String ordinal = switch (i) { case 1 -> "First"; case 2 -> "Second"; case 3 -> "Third"; default -> "Value too big"; }; System.out.println(ordinal);</pre>	<p data-bbox="1828 852 2383 919">Second</p>

Allowed types for `switch` statements

- Integer types, related by `boxing / unboxing`:
 - `byte` and `Byte`
 - `int` and `Integer`
 - `short` and `Short`
 - `char` and `Character`
- `String`
- `enum` types

Allowed labels

```
int wed = 3;

int number = 6;

switch(number) {
  case 1      -> System.out.println("Monday"); // o.k., constant int literal
  case 1 + 1 -> System.out.println("Tuesday"); // o.k. constant int expression
  case wed   -> System.out.println("Wednesday"); // Error: Constant expression required
  ...
}
```

Overview

Statements

↳ Loops

Why loops?

Objective: Execute the same statement multiple times.

“Solution”: Copy / paste the statement in question:

```
System.out.println("Do not copy!");  
System.out.println("Do not copy!");  
System.out.println("Do not copy!");  
System.out.println("Do not copy!");
```

Problem: Desired number of repetitions must be known at compile time.

Number of repetitions given by user input

```
System.out.print("Enter desired number of repetitions: ");  
final int repetitions = scan.nextInt();  
  
switch(repetitions) { // Employing fall-through  
    case 5: System.out.println("Do not copy!");  
    case 4: System.out.println("Do not copy!");  
    case 3: System.out.println("Do not copy!");  
    case 2: System.out.println("Do not copy!");  
    case 1: System.out.println("Do not copy!");  
}
```

Limited and clumsy workaround.

Overview

Statements

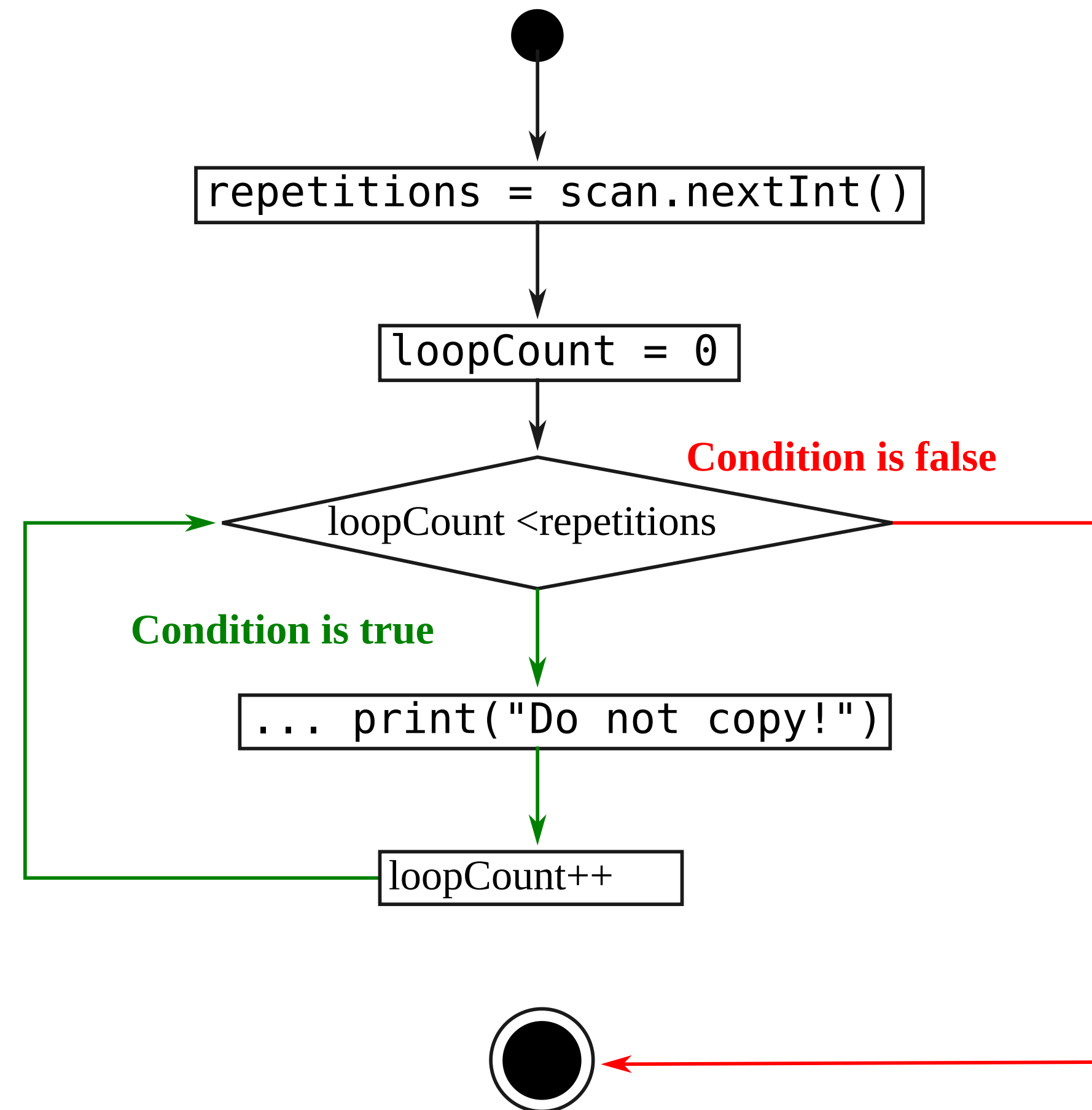
↳ Loops

↳ while

A while loop

```
final int repetitions = scan.nextInt(); ❶  
int loopCount = 0; ❷  
  
while (loopCount < repetitions ❸) {  
    System.out.println("Do not copy!"); ❹  
    loopCount++; ❺  
}
```

```
Do not copy!  
Do not copy!  
Do not copy!
```



Combining increment and termination condition

Code	Execution
<pre>System.out.print("Enter repetitions: "); final int repetitions = scan.nextInt(); int loopCounter = 0; while (loopCounter++ < repetitions) { System.out.println("Do not copy!"); }</pre>	<pre>Enter repetitions: 3 Do not copy! Do not copy! Do not copy!</pre>

while syntax

```
while (booleanExpression)  
  (block | statement)
```

Empty `while` body

```
int threeSeries = 1;  
while ((threeSeries *=3 ) < 100);  
System.out.println(threeSeries);
```

Exercise: Guess resulting output.

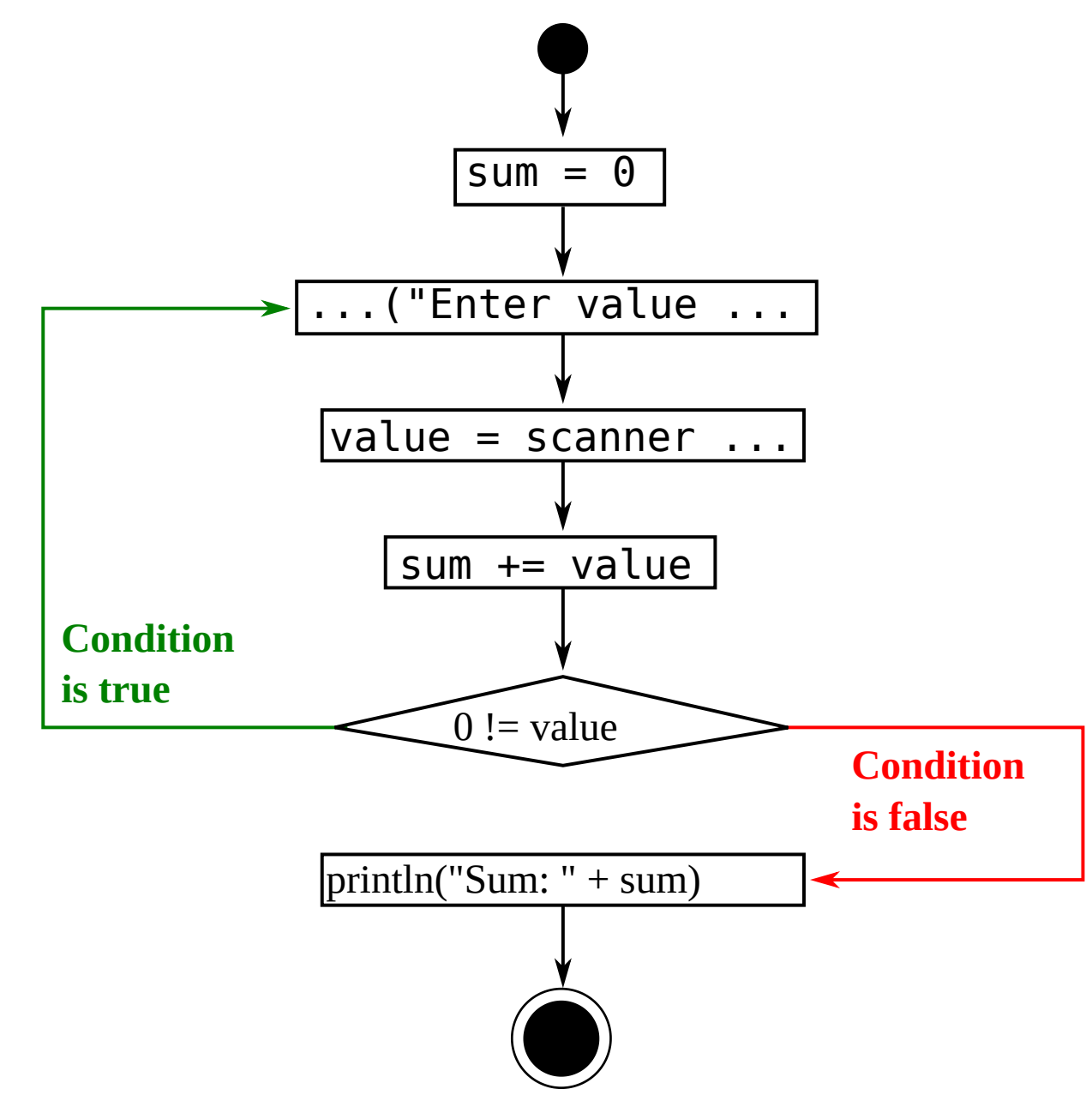
Followup exercises

- 68. Generating square numbers
- 69. Calculating factorial

A do ... while loop

```
int sum = 0, value;  
do {  
    System.out.print(  
        "Enter value, 0 to terminate: ");  
    value = scan.nextInt();  
    sum += value;  
} while (0 != value);  
System.out.println("Sum: " + sum);
```

```
Enter value, 0 to terminate: 3  
Enter value, 0 to terminate: 1  
Enter value, 0 to terminate: 0  
Sum: 4
```



do ... while syntax

```
do  
  (block | statement)  
while (booleanExpression);
```

Followup exercises

70. Even or odd?

71. Square root approximation

Overview

Statements

⇒ Loops

⇒ for

Frequent usage of `while`

```
int i = 0; ①  
while (i < 5 ②) {  
    ...  
    i++; ③  
}
```

- ① Declaring and initializing a loop termination variable.
- ② Check for loop termination.
- ③ Loop progression control

Nice to have: **More concise syntax**

Replacing `while(...)` by `for(...;...;...)`

```
for (int i = 0 ①; i < 5 ②; i++ ③) {  
    ...  
}
```

```
int i = 0; ①  
while (i < 5 ②) {  
    ...  
    i++; ③  
}
```

for syntax

```
for ( init ; booleanExpression ; update )  
  (block | statement)
```

for variable scope

```
for (int i = 0 ; i < 3; i++){ // i being defined inside loop's sco  
    System.out.println(i);  
}  
System.out.println(i);      // Error: i undefined outside loop's bo
```

```
int i; // i being defined in parent sco  
for (i = 0; i < 3; i++) {  
    System.out.println(i);  
}  
System.out.println(i); // o.K.
```


for variable scope equivalence

```
for (int i = 0 ; i < 3; i++) {  
    System.out.println(i);  
}  
System.out.println(i);           // Error: i undefined in parent scope
```

```
{ // Beginning of block  
    int i = 0;  
    for (; i < 3; i++) {  
        System.out.println(i);  
    }  
} // Ending of block  
System.out.println(i);           // Error: i undefined in parent scope
```

for vs. while relationship

```
while ( expression )  
  (block | statement)
```

```
for (... ;expression ; ...)  
  (block | statement)
```

Observation: `for (... ; ... ; ...)` is more general than `while(...)`.

Followup exercises

72. Printing numbers

73. Printing just even numbers

Nested loops 1

```
for (int i = 1; i <= 2; i++) {  
    for (int j = 1; j <= 3; j++) {  
        System.out.print("(" + i + "|" + j + ") ");  
    }  
    System.out.println(); // newline  
}
```

```
(1|1) (1|2) (1|3)  
(2|1) (2|2) (2|3)
```

Nested loops 2

```
for (int i = 0; i < 6; i++) {  
    for (int j = 0; j < i; j++) {  
        System.out.print(i + j + " ");  
    }  
    System.out.println(); // newline  
}
```

```
1  
2 3  
3 4 5  
4 5 6 7  
5 6 7 8 9
```

Better readability: **row** and **column** in favour of **i** and **j**

```
// What do i and j actually
// represent?

for (int i = 0; i < 6; i++) {
    for (int j = 0; j < i; j++) {
        System.out.print(i + j + " ");
    }
    System.out.println();
}
```

```
// Improved code readability.

for (int row = 0; row < 6; row++) {
    for (int column = 0;
        column < row; column++) {
        System.out.print(
            row + column + " ");
    }
    System.out.println();
}
```

Followup exercises

74. Merry Xmas
75. More fun with Xmas trees
76. A basic square number table
77. Tidy up the mess!
78. HTML-ify me
79. Auxiliary Example, part 1: A multiplication table
80. Auxiliary Example, part 2: Avoiding redundant entries
81. Creating a “real” square table
82. Creating a sophisticated HTML version of your square table

Overview

Statements

↳ Loops

↳ for

↳ Loops and calculations

Calculating values

```
final int LIMIT = 5;  
int sum = 0;  
  
for (int i = 1; i <= LIMIT; i++) {  
    sum += i;  
}  
  
System.out.println("1 + ... + " + LIMIT + " = " + sum);
```

```
1 + ... + 5 = 15
```

Followup exercises

- 83. Display all summands
- 84. Playing lottery
- 85. Guessing numbers
- 86. Smallest multiple
- 87. Smallest multiple, purely algebraic solution
- 88. Pythagorean triples
- 89. Avoiding duplicates and gaining performance

Statements

↳ Using automated tests.

Response to coding errors

Given a day of the week encoded as 0=Sun, 1=Mon, 2=Tue, ...6=Sat, and a boolean indicating if we are on vacation, return a string of the form "7:00" indicating when the alarm clock should ring. Weekdays, the alarm should be "7:00" and on the weekend it should be "10:00". Unless we are on vacation -- then on weekdays it should be "10:00" and weekends it should be "off".

alarmClock(1, false) → "7:00"
alarmClock(5, false) → "7:00"
alarmClock(0, false) → "10:00"

Go

...Save, Compile, Run (ctrl-enter)

```
public String alarmClock(int day, boolean vacation) {  
    switch(day) {  
        case 1:  
        case 2:  
        case 3:  
        case 4:  
            return vacation? "10:00" : "7:00";  
    }  
    return vacation? "off" : "10:00";  
}
```

Expected	Run		
alarmClock(1, false) → "7:00"	"7:00"	OK	Green
alarmClock(5, false) → "7:00"	"10:00"	X	Red
alarmClock(0, false) → "10:00"	"10:00"	OK	Green
alarmClock(6, false) → "10:00"	"10:00"	OK	Green
alarmClock(0, true) → "off"	"off"	OK	Green
alarmClock(6, true) → "off"	"off"	OK	Green
alarmClock(1, true) → "10:00"	"10:00"	OK	Green
alarmClock(3, true) → "10:00"	"10:00"	OK	Green
alarmClock(5, true) → "10:00"	"off"	X	Red
other tests		OK	Green

Unit test concept

- Will be [explained in detail](#).
- Idea: Feed in samples, check results for correctness.
- Previous slide: [Logic-1 > alarmClock](#)
- Sample project at [MI Gitlab](#).

alarmClock(...) with errors

```
public class AlarmClock {  
    /** Given a day of the week encoded as 0=Sun, 1=Mon,...  
     */  
    static ❶ public String alarmClock(int day, boolean vacation) {  
        switch (day) {  
            case 1:  
  
                ...  
            if (vacation) {  
                return "off";  
            } else {  
                return "10:00"; ...  
            }  
        }  
    }  
}
```

Testing alarmClock(...)

```
public class AlarmClockTest {  
    @Test ❶  
    public void test_1_false() {  
        Assert.assertEquals( "7:00", AlarmClock.alarmClock(1, false));  
    }  
    ...  
  
    @Test  
    public void test_0_false() {  
        Assert.assertEquals("10:00", AlarmClock.alarmClock(0, false));  
    }  
    ...  
}
```

Expected result

Input parameter

Testing alarmClock(...) details

```
public class AlarmClockTest {  
    @Test  
    public void test_1_false() {  
        final String result = AlarmClock.alarmClock(1, false);  
  
        Assert.assertEquals( "7:00", result);  
    }  
    ...  
}
```

Input parameter

Expected result