# 10 SELECTIONS

Terry Marris 19 April 2001

## 10.1 OBJECTIVES

By the end of this lesson the student should be able to

- understand and use 2-way (if ... else ...) selection statements
- understand and use multi-way (if .. else if ... ) selection statements
- understand and use the logical operators and, or and not .

## 10.2 PRE-REQUISITES

The student should be comfortable with boolean expressions, dry runs, boundary testing and test plans, the *int* data type and simple arithmetic expressions.

## 10.3 PREVIEW

We see how to make a selection between alternative courses of action. We introduce the *if* and *else* keywords. We introduce the logical operators && (and) and || (or).

## 10.4 if ... else ...

*if* and *else* are Java keywords. We use them to make a selection between alternatives. If some condition is met, then do this, else (otherwise the condition is not met) do that.

For example, an account must always remain in credit (i.e. balance >=0). A transaction amount is to be paid from the account. *If* there is enough funds in the account then deduct the transaction from the balance *else* leave the balance unchanged.

```
if (balance - transaction >= 0) {
  isEnoughFunds = true;
  balance = balance - transaction;
}
else {
  isEnoughFunds = false;
}
```

If subtracting the *transaction* amount from the *balance* results in a value greater than (or equal to) zero, assign *true* to *isEnoughFunds* and deduct the *transaction* amount from the *balance*, otherwise assign *false* to *isEnoughFunds*.

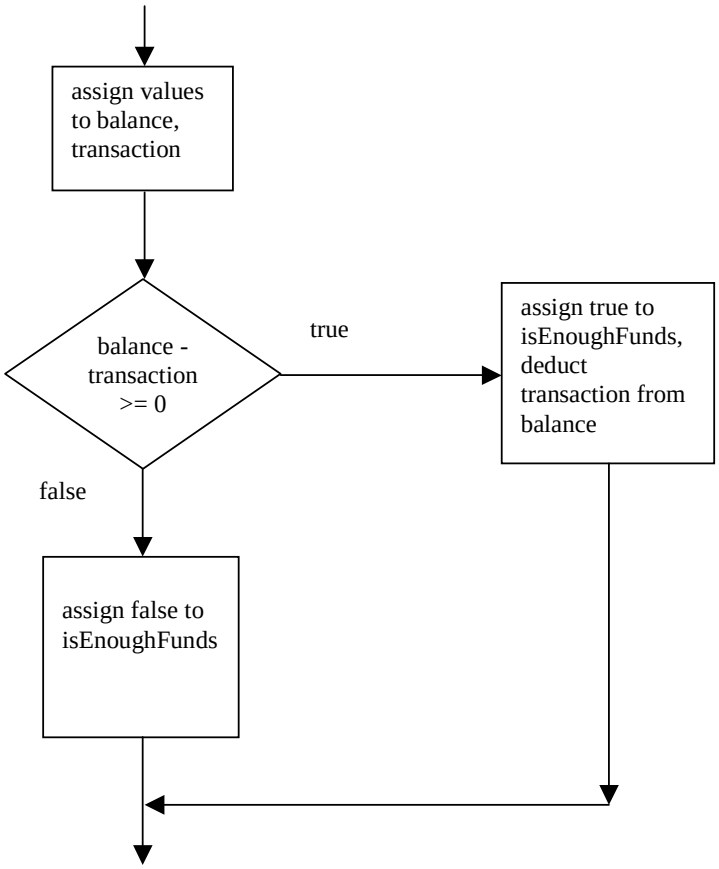



**Figure 10.1** *if ... else ... flow of control*

The braces, { and }, group statements into blocks. The block

```
  isEnoughFunds = true;
  balance = balance - transaction;
```

is executed only if the boolean expression *balance - transaction >= 0* is true.

The block

```
isEnoughFunds = false;
```

is executed only if the expression *balance - transaction >= 0* is false.

Braces are not explicitly required if there is just one statement in the block, as in the *else* clause.

```
if (balance - transaction >= 0) {
  isEnoughFunds = true;
  balance = balance - transaction;
}
else
  isEnoughFunds = false;
```

The *else* clause is not compulsory, it depends on the logic.

```
isEnoughFunds = false;
if (balance - transaction >= 0) {
  isEnoughFunds = true;
  balance = balance - transaction;
}
```

As a matter of layout style notice

- the positioning of the braces - *{* on same line as *if* and *else*, *}* on its own line below and vertically below *if* and *else*
- *else* is aligned vertically beneath the *if*.
- statement blocks are indented under the *if* and *else* keywords

Semi-colons, the statement terminators, do not usually follow an *if* or an *else*.

## 10.5 if ... else if ...

*if ... else ...* introduced in §12.4 above selects one from two alternatives. *if ... else if ... else ...* selects one from several alternatives.

The Java code fragment shown below assigns *loss made* or *break even* or *in profit* depending on the whether *income* is less than, equal to or more than *expenditure*.

```
if (income < expenditure)
  state = "loss made";
else if (income == expenditure)
  state = "break even";
else if (income > expenditure)
  state = "in profit";
```

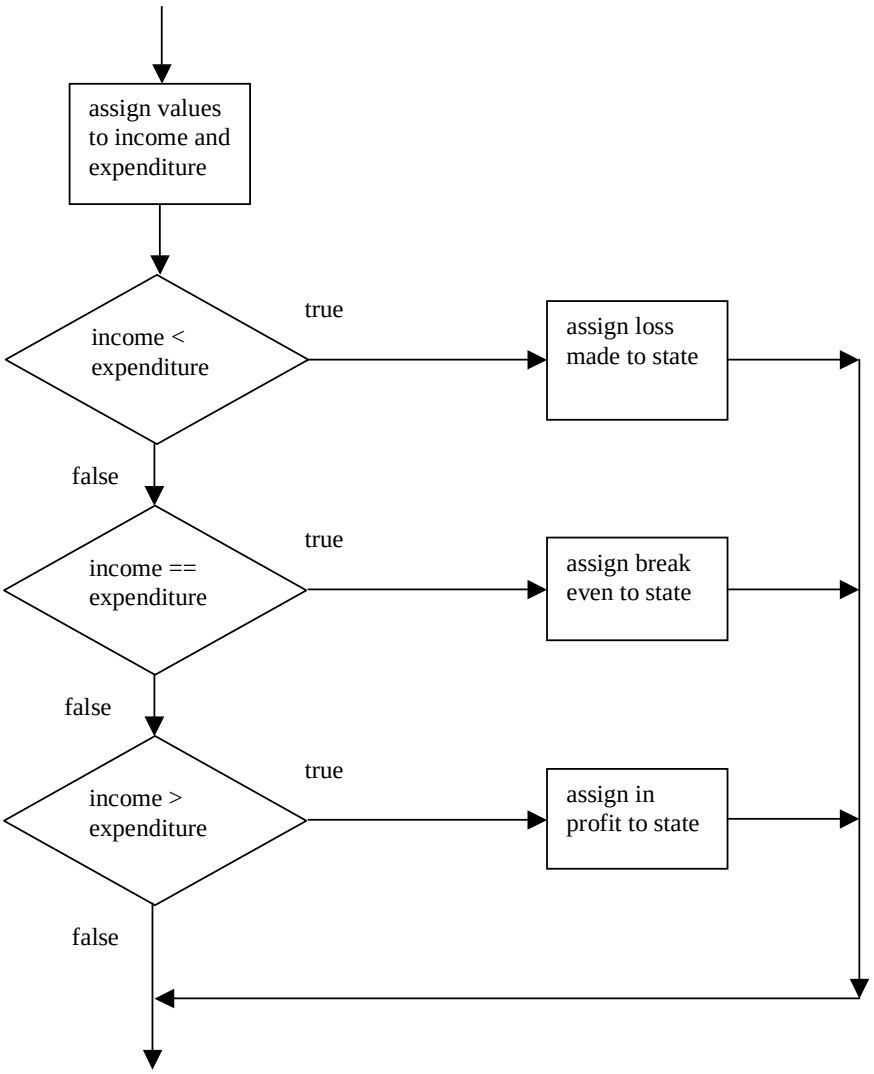



**Figure 10.2** *if ... else if ... else if ... flow of control*

Understanding a sequence of *if ... else if*s is easy. Just

- look down the sequence of boolean expressions
- find the first one that is true
- execute the corresponding action
- skip to the end

For example, a college grades students' work as Refer, Pass, Merit or Distinction depending on the percentage mark awarded like this

| Grade | Refer | Pass | Merit | Distinction |
|---|---|---|---|---|
| Mark | 0..39 | 40..54 | 55..69 | 70..100 |

In Java we could write

```
if (mark < 0)
  grade = "error - mark is less than zero";
else if (mark < 40)
  grade = "refer";
else if (mark < 55)
  grade = "pass";
else if (mark < 70)
  grade = "merit")
else if (mark <= 100)
  grade = "distinction";
else
  grade = "error - mark exceeds 100%";
```

Suppose *mark* has the value 56.

Look down the sequence of boolean expressions *(mark < 0)*, *(mark < 40)*, *(mark < 55)*, *(mark < 70* and *(mark <= 100)*.

*(mark < 0)* is false. *(mark < 40)* is false. *(mark < 55)* is false.

The first one that is true is *(mark < 70)*.

Therefore *grade = "merit"*.

As before, we can group blocks of statements between { and } if required. We can do without the trailing else if not required to capture "none of the above" cases.

Also notice the layout. The *else if* ... are lined up directly under the opening *if* (otherwise, if *else ifs* ... are indented one under the other, the code could end up marching across and off the page).

## 10.6 NESTED SELECTIONS

We can have *if*s within *if*s.

In the absence of braces, an *else* is always matched with the closest previous *if*.

```
int n = 0;
int a = 1;
int b = 2;
int z = -1;
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

leaves *z* with value -1 (since *n* is not more than zero, $a > b$ is not tested).

```
int n = 0;
int a = 1;
int b = 2;
int z = -1;
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

leaves *z* with value 2 (since *n* is not more than zero).

## 10.7 LOGICAL AND

The Java operator for logical and is *&&*. It means *and at the same time*. So the Java code fragment

```
if (mark >= 0 && mark < 40)
  grade = "refer";
```

says if *mark* is greater than or equal to zero *and at the same time* if *mark* is less than 40, then *grade* is *refer*.

The Java code fragment shown below might be used for controlling the laser in a compact disk player. To prevent users from burning themselves with the laser beam, it is lit up only if the door is closed and, at the same time, the power switch is on. If either the door is not closed, or the switch is not on, the laser remains unlit.

```
if (doorIsClosed && switchIsOn) {
  startUpMotor();
  lightUpLaser();
}
```

| doorIsClosed | switchIsOn | startUpMotor() | lightUpLaser() |
|---|---|---|---|
| true | true | ✓ | ✓ |
| true | false | ✗ | ✗ |
| false | true | ✗ | ✗ |
| false | false | ✗ | ✗ |

**Figure 10.3** *both doorIsClosed and switchIsOn must be true before the laser is lit up*

## 10.8 LOGICAL OR

The Java operator for logical or is ||. It means *or*. So the Java code fragment

```
if (mark < 0 || mark > 100)
  grade = "error - out of range 0..100";
```

says if the *mark* is less than zero or if *mark* is more than 100, then *grade* is *error - out of range 0..100*.

Look at the Java code fragment shown below. Could it be used to prevent a user from burning themselves with the laser beam?

```
if (doorIsClosed || switchIsOn) {
  startUpMotor();
  lightUpLaser();
}
```

| doorIsClosed | switchIsOn | startUpMotor() | lightUpLaser() |
|---|---|---|---|
| true | true | ✓ | ✓ |
| true | false | ✓ | ✓ |
| false | true | ✓ | ✓ |
| false | false | ✕ | ✕ |

**Figure 10.4** laser is lit up if either *doorIsClosed or switchIsOn*

Evidently not. Either doorIsClosed or switchIsOn or both are true for the laser to be lit up.

## 10.9 LOGICAL NOT

We have already met the operator *!=* (not equals). The Java operator for logical not is the exclamation mark, *!*.

This code fragment

```
if (x != 0)
  y = z / x;
```

prevents division by zero.

## 10.10 PRECEDENCE

The logical and or operators come between the == and = operators according to the rules of precedence, with && having a higher precedence than ||.

| Operators | | | | | | | | Associativity |
|---|---|---|---|---|---|---|---|---|
| . | () (method call) | | | | | | | left to right |
| ! | ++ | -- | + (unary) | - (unary) | () | (cast) | new | right to left |
| * | / | % | | | | | | left to right |
| + | - | | | | | | | left to right |
| < | <= | > | >= | | | | | left to right |
| == | != | | | | | | | left to right |
| && | | | | | | | | left to right |
| \|\| | | | | | | | | left to right |
| = | | | | | | | | right to left |

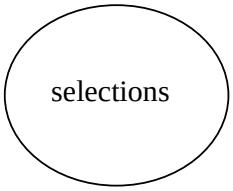
**Figure 10.5** *Operator Precedence*

If no brackets are used, operations are performed in the order given in the table, those higher up in the table being processed before those lower down. Operators on the same level are processed from left to right, unless they associate from right to left.

## 10.11 FURTHER READING

LEWIS & LOFTHOUSE Java Software Solutions pp111-125
ARNOW & WEIS Java - An Object Oriented Approach pp190-226
HORSTMAN N & CORNELL core Java 2 Volume 1 Fundamentals pp 57,59, 75-78,

**10.12 REVIEW**




if ... else

```
if (age >= 17)
  application = "approved"
else
  application = "rejected"
```

if ... else if ...

```
if (chest <= 36)
  size = "small";
else if (chest <= 40)
  size = "medium";
else if (chest <= 44)
  size = "large";
else
  size = "extra large";
```

&& - and at the same time

```
if (chest > 36 && chest <= 40)
  size = "medium";
```

|| - or

```
if (chest < 32 || chest > 48)
  size = "unknown";
```

## 10.13 EXERCISES

**1** Explain, with the aid of appropriate examples, the meaning of

**(a)** && **(b)** || **(c)** !

**2** Dry run each of the following Java code fragments:

**(a)**
```
int n = 0;
int a = 1;
int b = 1;
int z = -1;
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

**(b)**
```
int n = 1;
int a = 1;
int b = 1;
int z = -1;
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

**(c)**
```
int n = 1;
int a = 2;
int b = 1;
int z = -1;
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

**(d)**
```
int n = 0;
int a = 2;
int b = 1;
int x = -1;
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

**3** Dry run each of the following Java code fragments:

**(a)**
```
int n = 0;
int a = 1;
int b = 1;
int z = -1;
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

**(b)**
```
int n = 1;
int a = 1;
int b = 1;
int z = -1;
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

**(c)**
```
int n = 1;
int a = 2;
int b = 1;
int z = -1;
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

**(d)**
```
int n = 0;
int a = 2;
int b = 1;
int x = -1;
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

**4** Dry run the following code fragment three times: **(a)** when target = 3, **(b)** when target = 4, **(c)** when target = 5

```
boolean isFound = false;
int lo = 0;
int hi = 9;
int mid = (lo + hi) / 2;
if (target == mid)
  isFound = true;
else if (target < mid)
  hi = mid - 1;
else if (target > mid)
  lo = mid + 1;
```

**5** Write a Java code fragment to reflect the following scenario:

*Air quality* is *pleasant* if the *pollution index* is less than 35, *unpleasant* if the *index* is 35 up to 60 inclusive, and *dangerous* if the *index* is above 60.

**6** Devise a test plan for the following code fragments

**(a)**
```
if (a != 0 && x >= 0)
 b = true;
else
  b = false;
```

**(b)**
```
if (a != 0 || x >= 0)
  b = true
else
  b = false;
```

**7** flip and flop are both boolean variables. Devise a test plan for the following code fragment:

```
if (flip) {
  flop = true;
  flip = false
}
else if (flop) {
  flop = false;
  flip = true;
}
```