

Control Structure

Sang Shin
JPassion.com

“Code with Passion!”



Topics

- Decision control structures (if, else, switch)
- Repetition control structures (while, do-while, for)
- Branching statements (break, continue, return)

Decision Control Structures

Decision Control Structures

- Decision control structures
 - > Java statements that allows us to select and execute specific blocks of code while skipping other sections
- Types:
 - > if-statement
 - > if-else-statement
 - > If-else if-statement

if-statement


- if-statement
 - > Specifies that a statement (or statements in a block of code) will be executed if and only if a boolean condition is true

- if-statement has the form:

```
if( boolean_expression )  
    statement;
```

or

```
if( boolean_expression ){  
    statement1;  
    statement2;  
}
```



Recommended even for
a single statement

- > `boolean_expression` is either a boolean expression or boolean variable.

Examples: if statement

```
// Example #1
int grade = 68;
if( grade > 60 )
    System.out.println("Congratulations!");
```

```
// Example #2
int grade = 68;
if( grade > 60 ){
    System.out.println("Congratulations!");
    System.out.println("You passed!");
}
```

if-else statement

- if-else statement
 - > Used when we want to execute a block of code if a boolean condition is true, and a different block of code otherwise
- if-else statement has the form:

```
if( boolean_expression ){
    statement1;
    statement2;
    . . .
}
else{
    statement3;
    statement4;
    . . .
}
```

if-else-if statement

- The statement has the form:

```
if( boolean_expression1 ){
    statement1;
}
else if( boolean_expression2 ){
    statement2;
}
else {
    statement3;
}
```


Example: if-else-if statement

```
int grade = 68;

if ( grade > 90 ){
    System.out.println("Very good!");
}
else if ( grade > 60 ){
    System.out.println("Good!");
}
else{
    System.out.println("Sorry you failed");
}
```

Common Errors

1. The condition inside the if-statement does not evaluate to a boolean value. For example,

```
// Compile error because number is not boolean
int number = 0;
if( number ){
    //some statements here
}
```

2. Writing **elseif** instead of **else if**

Common Errors

3. Using = instead of == for comparison. For example,

```
//WRONG
int number = 0;
if( number = 0 ){
    //some statements here
}
```

This should be written as,

```
//CORRECT
int number = 0;
if( number == 0 ){
    //some statements here
}
```

switch-statement

- switch
 - > Allows branching on multiple outcomes
- switch statement has the form:

```
switch( switch_expression ) {
    case case_selector1:
        statement1; //
        statement2; //block 1
        break;
    case case_selector2:
        statement1; //
        statement2; //block 2
        break;
        :
    default:
        statement1; //
        statement2; //block n
}
```

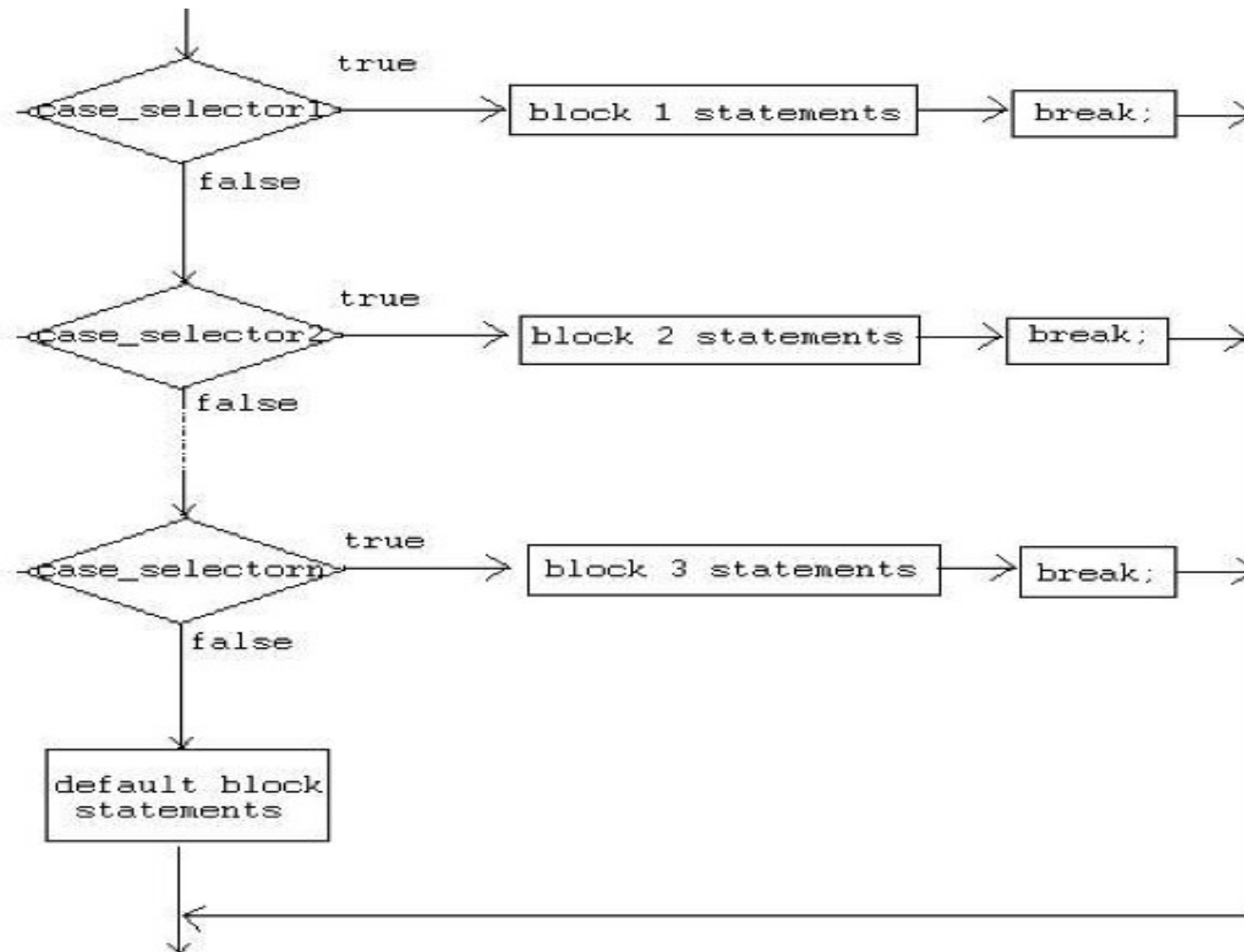
switch-statement

- where,
 - > switch_expression
 - > Integer or character expression
 - > String expression – only from Java SE 7
 - > case_selector1, case_selector2 and so on,
 - > Are unique integer or character constants.

switch-statement

- When a switch is encountered,
 - > Java first evaluates the `switch_expression`, and jumps to the case whose selector matches the value of the expression.
 - > The program executes the statements in order from that point on until a `break` statement is encountered
 - > If none of the cases are satisfied, the default block is executed
- Unlike with the `if` statement, the multiple statements are executed in the `switch` statement without needing the curly braces.
- When a case in a `switch` statement has been matched, all the statements associated with that case are executed

Flowchart



Example: switch statement

```
1  public class Grade {
2      public static void main( String[] args ) {
3          int grade = 92;
4          switch ((grade/10)*10) { // Round down to nearest 10
5              case 100:
6                  System.out.println( "Excellent!" );
7                  break;
8              case 90:
9                  System.out.println("Good job!" );
10                 break;
11             case 80:
12                 System.out.println("Study harder!" );
13                 break;
14             default: // If there is no match, this will be chosen
15                 System.out.println("Sorry, you failed.");
16             }
17         }
18     }
```


Coding Guidelines

- Deciding whether to use an if statement or a switch statement is a judgment call
 - > You can decide which to use, based on readability and other factors.
- An if statement can be used to make decisions based on ranges of values or conditions, whereas a switch statement can make decisions based only on a single value. Also, the value provided to each case statement must be unique.

Lab:

Exercise 1: “if/else” control structure
1034_javase_control.zip



Repetition Control Structures

Repetition Control Structures

- Repetition control structures
 - > Are Java statements that allows us to execute specific blocks of code a number of times.
- Types:
 - > while-loop
 - > do-while loop
 - > for-loop

while-loop

- while loop
 - > is a statement or block of code that is repeated as long as some condition is satisfied.

- while loop has the form:

```
while( boolean_expression ){  
    statement1;  
    statement2;  
    . . .  
}
```

- > The statements inside the while loop are executed as long as the `boolean_expression` evaluates to true.

Examples: while loop

// Example #1

```
int x = 0;
```

```
while (x<10) {  
    System.out.println(x);  
    x++;  
}
```

// Example #2: infinite loop

```
while(true)  
    System.out.println("hello");
```

// Example #3: no loops

// statement is not even executed

```
while (false)  
    System.out.println("hello");
```

do-while-loop

- do-while loop
 - > is similar to the while-loop
 - > statements inside a do-while loop are executed several times as long as the condition is satisfied
 - > The main difference between a while and do-while loop:
 - > the statements inside a do-while loop are executed at least once.

- do-while loop has the form:

```
do{  
    statement1;  
    statement2;  
}while( boolean_expression );
```

Examples: do-while loop

// Example #1

```
int x = 0;
do {
    System.out.println(x);
    x++;
}while (x<10);
```

// Example #2: infinite loop

```
do{
    System.out.println("hello");
} while (true);
```

// Example #3: one loop

// statement is executed once

```
do
    System.out.println("hello");
while (false);
```


for-loop

- for loop

- > allows execution of the same code a number of times.

- for loop has the form (pre-Java SE 5 format)

```
for(InitializationExpression; LoopCondition; StepExpression) {  
    statement1;  
    statement2;  
    . . .  
}
```

- > where,

- > InitializationExpression - initializes the loop variable.

- > LoopCondition - compares the loop variable to some limit value.

- > StepExpression - updates the loop variable.

Example: for loop

```
int i;  
for( i = 0; i < 10; i++){  
    System.out.println(i);  
}
```

- The code shown above is equivalent to the following while loop.

```
int i = 0;  
while( i < 10 ){  
    System.out.print(i);  
    i++;  
}
```

Lab:

Exercise 2: “for” loop

Exercise 3: “while” loop

1034_javase_control.zip



Branching Statements

Branching Statements

- Branching statements allows us to redirect the flow of program execution.
- Java offers three branching statements:
 - > break
 - > continue
 - > return

“break” statement

- Terminates the enclosing switch statement, and flow of control transfers to the statement immediately following the switch.
- This can also be used to terminate a for, while, or do-while loop

Example: “break” statement

```
String names[]={"Beah","Bianca","Lance","Belle","Nico","Yza","Gem","Ethan"};

String searchName = "Yza";
boolean foundName = false;

for( int i=0; i< names.length; i++ ){
    if( names[i].equals( searchName )){
        foundName = true;
        break;
    }
}

if( foundName ) System.out.println( searchName + " found!" );
else System.out.println( searchName + " not found." );
```

“continue” statement

- Skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop, basically skipping the remainder of this iteration of the loop.

Example: “continue”

```
String names[] = {"Beah", "Bianca", "Lance", "Beah"};
int    count = 0;

for( int i=0; i<names.length; i++ ){
    if( !names[i].equals("Beah") ){
        continue; //skip next statement
    }
    count++;
}

System.out.println("There are "+count+" Beahs in the list");
```

return statement

- Used to exit from current method
 - > Flow of control returns to the statement that follows the original method call.
- To return a value
 - > Simply put the value (or an expression that calculates the value) after the return keyword.
 - > For example,

```
return ++count;
```

or

```
return "Hello";
```
 - > The data type of the value returned by return must match the type of the method's declared return value.

Code with Passion!
JPassion.com

