DECISION CONTROL CONSTRUCTS IN JAVA

Decision control statements can change the execution flow of a program.

Decision control statements in Java are:

- if statement
- Conditional operator
- switch statement

If statement

The if statement is followed by a body block. The if statement evaluate logical expressions and make a decision whether to execute the block or not based on whether the logical expression returns a true or false.

Syntax: if

```
if(condition)

{//do something}
```

- If the condition is true the if block is executed, else not.
- The condition should be a boolean variable or a logical expression that will evaluate to boolean values true or false.
  - If we specify anything other than a boolean variable or an expression that doesn't evaluate to true or false, you will get compilation error.
  - Even if you pass a String variable with a value of true or false, you will still get a compilation error.
- Curly braces {} are not required when only one statement is there for the “if”, "else if" or “else” clause, **it is always a good practice** to use curly braces {} to enclose if body for better clarity, and to avoid accidental mistakes.

if, else-if, else

The **“if statement” block** can be followed by any number of optional **“else if statement” blocks** and one optional **else block**.

Syntax: if, else-if, else

```
 if (cond1) {

//do something
```

```
} else if (cond2) {
//do something
} else if (cond3) {
//do something
} else {
//do something
}
```

- Only one of the if or “else if” blocks will be executed for the first true condition.
- If no if or “else if” conditions are true, then the else block is executed (if one is available).

Avoiding accidental mistakes

Usage of curly braces {} for enclosing if body

Curly braces {} are not required when only one statement is there for the “if”, "else if" or “else” clause.

```
if(condition)
  statement1;
```

However, it is always a good practice to use curly braces {} to enclose if body for better clarity, and to avoid accidental mistakes like below:

```
if(condition)
  statement1;
  statement2;
else
  statement3;
```

- Here,
  - the if statement has only a single statement, statement1.
  - The second statement, statement2 is actually not part of "if" block and hence statement1 will end the if loop.

- This is followed by the else clause, which is now not part of any if block and hence you will get a compilation error.

- A more serious problem occurs when the extra statement is in the else block as no exception is thrown, but the statement will always get executed irrespective of the else block.

- Another problem with not using a block statement is the dangling else problem. In a complex nested if scenario, an else might be associated with a different if than what we intended. The else keyword is paired with the closest if.

Using the assignment operator = instead of equality operator ==

1. If you use the assignment operator = instead of equality operator ==, an assignment will happen instead of evaluation.

2. When you say if(x=y), y is assigned to x and x is evaluated.

- If x and y are boolean, value of x is overwritten with the value of y and then x is evaluated.

- If the types compared are not boolean, you will get a compile time error as you are evaluating something which is not a boolean.

Example: assignment operator = instead of equality operator ==

```
boolean isCompleted=false;

if(isCompleted=true){

  System.out.println("Executed!!!");

}
```

- This will print "Executed!!!"

- Here, isCompleted=true will assign true to isCompleted overwriting its current value and then isCompleted is evaluated; hence the evaluation will always return true irrespective of the value of isCompleted.

- The true is actually not necessary in this case and we could have just used if(isCompleted) and avoided the above mistake.

Conditional operator

The conditional operator can be considered as a limited form of "if" statement.

It is also called the ternary operator as it has three components.

Syntax: conditional operator
LogicalExpression **?** ThenExpression **:** ElseExpression

- If the LogicalExpression evaluates to true, then the result of the ThenExpression is returned. Otherwise the result of the ElseExpression is returned.

Example:conditional operator

```
int a = 1;

int b = 5;

int r;

r= a>b ?a : b;
System.out.println(r);
```

- This will print 5.
  - Here the expression (a>b) evaluate to false and hence the value of b is assigned to r.
- **[Tip]** The use of the conditional operator is discouraged over “if-else“ due to its readability issues.

Switch statement

The **switch statement** provide multi-branch selections based upon **integer**, **enumeration**, or **String**expression.

Syntax: switch statement

```
switch (expression) {

case value: statements;

break;

case value: statements;

break;

…

default: statements

}
```

Example:switch statement

```
int abc = 7;


switch (abc) {

case 1:
  System.out.println("ONE");

  break;

case 5:
  System.out.println("FIVE");

  break;

default:
  System.out.println("DEFAULT");

  break;

case 7:
  System.out.println("SEVEN");

  break;

}
```

Important properties of switch statement in Java

1. Any of **byte**, **char**, **short**, and **int** data types (and their wrapper classes) can be used with an integer switch statement; **long and double are not allowed.**
1. Corresponding wrapper classes such as Character, Byte, Short, and Integer are also allowed.
2. Prior to Java 7, only integer variables could be used with a switch statement.
2. In switch statement, the value of a constant expression is checked and then the control is passed to a case that matches the constant expression.
1. If no case matches the value of the expression, control is passed to the default clause, if present.
3. A **default can come in between** the cases in the switch statement and doesn't have to be always the last. But usually it is written in the end.
4. The **break keyword** is used to end the code sequence for a case, and to exit the switch statement.

1. **If there is no break** for a case, all the remaining cases (including default) will be executed until a break is found or the end of switch is reached.
2. **Break** is not always needed at the end of the default clause if default is the last case, but, it is usually included, especially if default is not the last case.
5. With a **String switch statement**,
1. a **null value** assigned to a string variable used within a switch statement will **lead to java.lang.NullPointerException exception**.
2. the comparison made within a switch statement is case-sensitive.

Source : http://javajee.com/decision-control-constructs-in-java