


ADVANCED OBJECT ORIENTED PROGRAMMING

Engr. Anees Ahmed Soomro

Assistant Professor

CS QUEST Nawabshah

<https://anees-soomro.neocities.org>

- 
- 1. Java selection**
 - 2. Loop control statements**
 - 3. For statement, While statement and do while statement**
 - 4. Nested for, while, do- while, working with all loops**

Java selection & Loop control statements

- **Selection Statements**
- **Switch Statement**
- **Iteration Statements**

Selection Statements

- Java provides selection statements that allow the program to choose between alternative actions during execution. The choice is based on criteria specified in the selection statement. These selection statements are

- simple if Statement

- if-else Statement

- else-if Statement

- switch Statement

Simple if Statement

- The simple if statement has the following syntax:

if(<conditional expression>) <statement>

- It is used to decide whether an action is to be performed or not, based on a condition.

- The condition is specified by <conditional expression> and the action to be performed is specified by <statement>.

- The semantics of the simple if statement are straightforward.

- The <conditional expression> is evaluated first.

- If its value is true, then <statement> (called the if block) is executed and execution continues with the rest of the program.

- If the value is false, then the if block is skipped and execution continues with the rest of the program.

- The semantics are illustrated by the activity diagram in [Figure 5.1a](#).

if-else Statement

The if-else statement has the following syntax:

```
if(<conditional expression>)
```

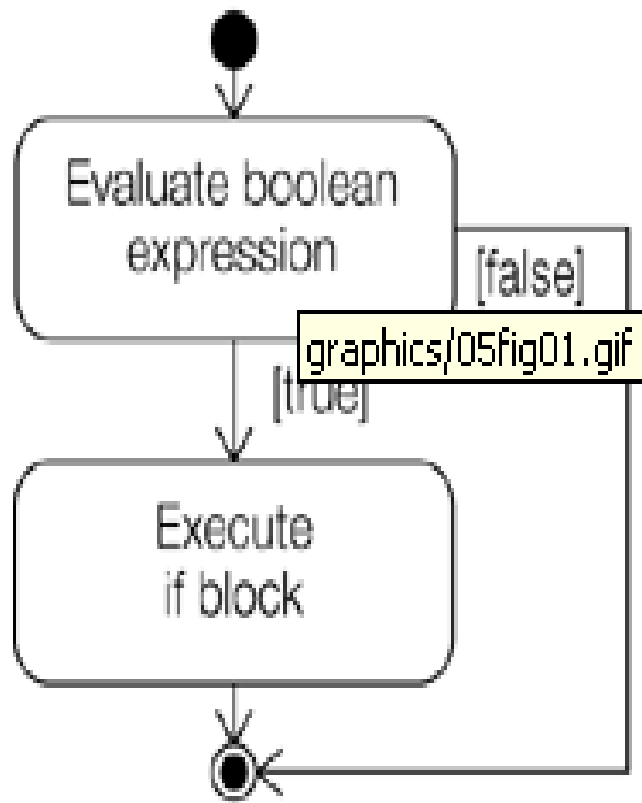
```
    <statement1>
```

```
else
```

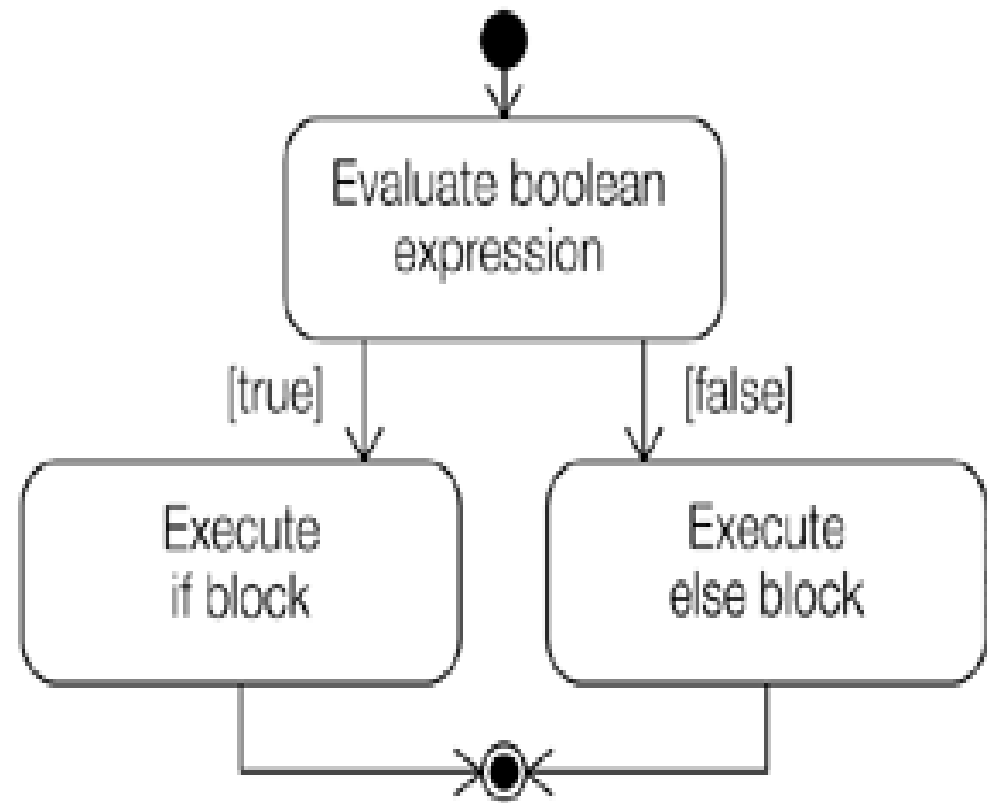
```
    <statement2>
```

- It is used to decide between two actions, based on a condition.
- The <conditional expression> is evaluated first. If its value is true, then <statement₁> (the if block) is executed and execution continues with the rest of the program.
- If the value is false, then <statement₂> (the else block) is executed and execution continues with the rest of the program.
- In other words, one of two mutually exclusive actions is performed.
- The else clause is optional; if omitted, the construct reduces to the simple if statement.
- The semantics are illustrated by the activity diagram in [Figure 5.1b](#).

Figure 5.1. Activity Diagram for **if** Statements



(a) Simple if Statement



(b) if-else Statement

else-if Statement

- The rule for matching an else clause is that an else clause always refers to the nearest if that is not already associated with another else clause.
- Block notation and proper indentation can be used to make the meaning obvious.
- Cascading if-else statements are a sequence of nested if-else statements where the if of the next if-else statement is joined to the else clause of the previous one.
- The decision to execute a block is then based on all the conditions evaluated so far.

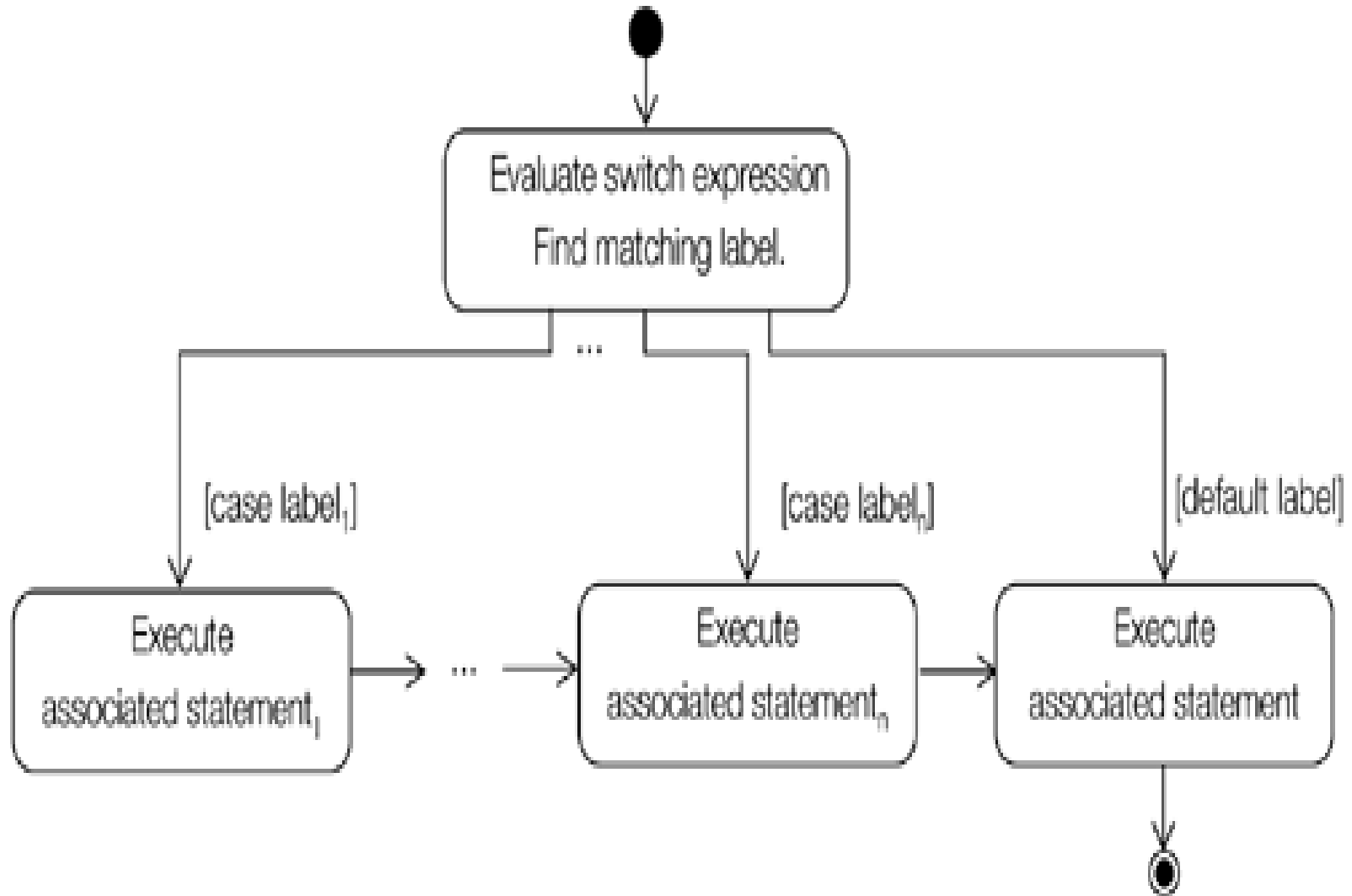
```
if (temperature >= upperLimit)
{
    // (1)  soundAlarm();  turnHeaterOff();}
else if (temperature < lowerLimit)
{ // (2) soundAlarm(); turnHeaterOn();}
else if (temperature == (upperLimit-lowerLimit)/2)
{ // (3)  doingFine();}
else
// (4)  noCauseToWorry();
```

switch Statement

- Conceptually the switch statement can be used to choose one among many alternative actions, based on the value of an expression. Its general form is as follows:
- ```
switch(<non-long integral expression>) {
 case label1: <statement1>
 case label2: <statement2>
 ...
 case labeln: <statementn>
 default: <statement>
} // end switch
```
- The syntax of the switch statement comprises a switch expression followed by the switch body, which is a block of statements. The type of the switch expression is non-long integral (i.e., char, byte, short, or int). The statements in the switch body can be labeled, defining entry points in the switch body where control can be transferred depending on the value of the switch expression. The semantics of the switch statement are as follows:
- The switch expression is evaluated first.
- The value of the switch expression is compared with the case labels. Control is transferred to the <statement<sub>i</sub>> associated with the case label that is equal to the value of the switch expression. After execution of the associated statement, control falls through to the next statement unless appropriate action is taken.
- If no case label is equal to the value of the switch expression, the statement associated with the default label is executed.



# Activity Diagram for switch Statement



# Iteration Statements

- Loops allow a block of statements to be executed repeatedly (i.e., iterated). A boolean condition (called the loop condition) is commonly used to determine when to terminate the loop.
- The statements executed in the loop constitute the loop body.
- The loop body can be a single statement or a block.

Java provides three language constructs for constructing loops:

## 1. while statement

## 2. do-while statement

## 3. for statement

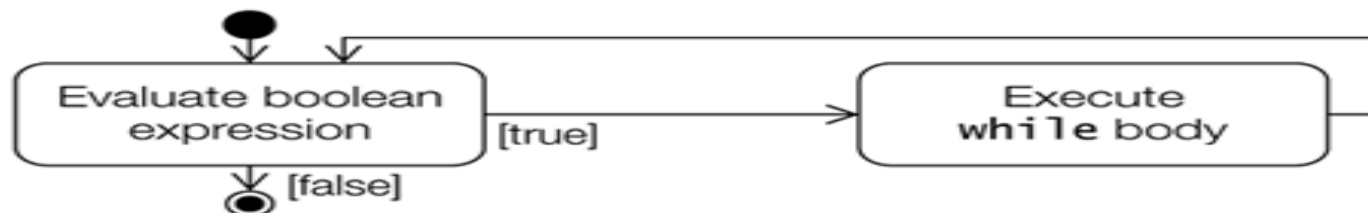
- These loops differ in the order in which they execute the loop body and test the loop condition.
- The while and the for loops test the loop condition before executing the loop body, while the do-while loop tests the loop condition after execution of the loop body.

# while Statement

The syntax of the while loop is

```
while(<loop condition>)
 <loop body>
```

- The loop condition is evaluated before executing the loop body. The while statement executes the loop body as long as the loop condition is true. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop. If the loop condition is false to begin with, the loop body is not executed at all. In other words, a while loop can execute zero or more times. The loop condition must be a boolean expression. The flow of control in a while statement is shown in figure below
- **Activity Diagram for while Statement**



# do-while Statement

- The syntax of the do-while loop is

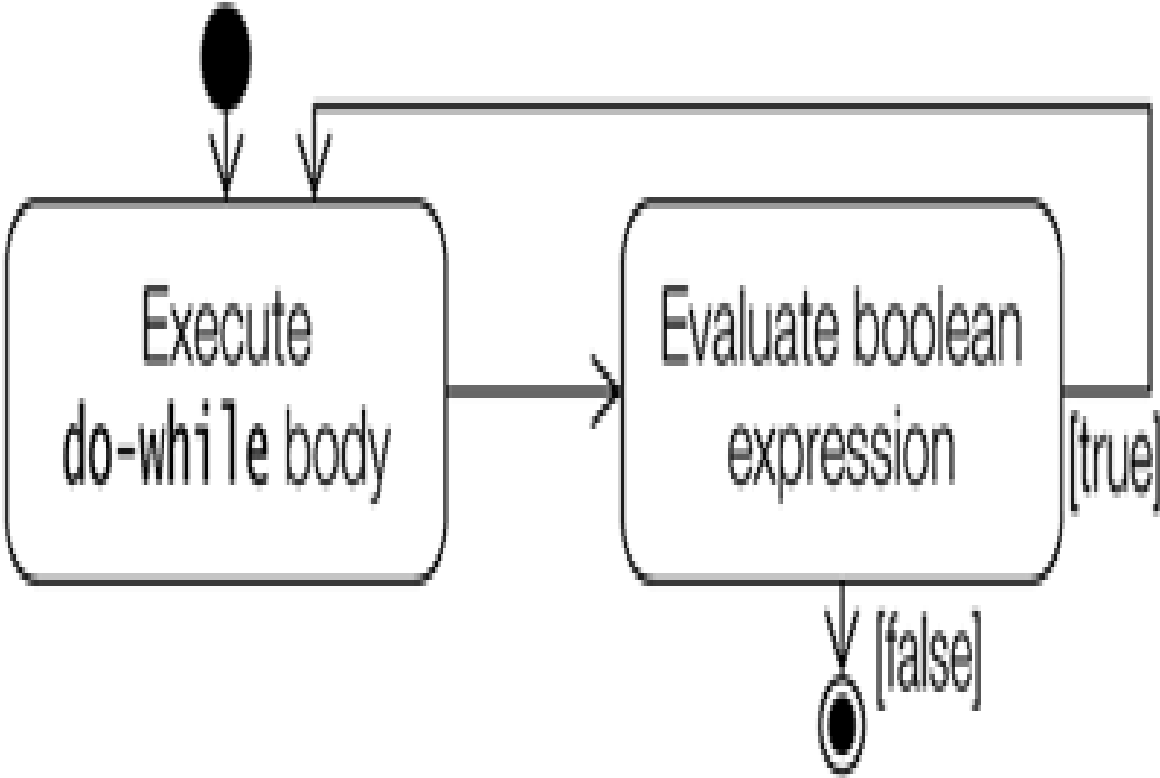
do

    <loop body>

while (<loop condition>);

- The loop condition is evaluated after executing the loop body. The do-while statement executes the loop body until the loop condition becomes false. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop. Note that the loop body is executed at least once. Figure in next slide illustrates the flow of control in a do-while statement.

# Activity Diagram for do-while Statement



# for Statement

- The for loop is the most general of all the loops. It is mostly used for counter-controlled loops, that is, when the number of iterations is known beforehand.
- The syntax of the loop is as follows:
- `for(<initialization>; <loop condition>; <increment expression>)  
    <loop body>`
- The <initialization> usually declares and initializes a loop variable that controls the execution of the <loop body>. The <loop condition> is a boolean expression, usually involving the loop variable, such that if the loop condition is true, the loop body is executed; otherwise, execution continues with the statement following the for loop. After each iteration (i.e., execution of the loop body), the <increment expression> is executed. This usually modifies the value of the loop variable to ensure eventual loop termination. The loop condition is then tested to determine if the loop body should be executed again. Note that the <initialization> is only executed once on entry to the loop. The semantics of the for loop are illustrated in Figure in next slide, and can be summarized by the following equivalent while loop code template:

# Activity Diagram for the for Statement

