FIT ICT Software Development

Lecture 4

Lecturer : Charles Tyner

RECAP……..

Three types of selection statements:

`if` statement:
Performs an action, if a condition is true; skips it, if false.
Single-selection statement—selects or ignores a single action (or group of actions).

`if…else` statement:
Performs an action if a condition is true and performs a different action if the condition is false.
Double-selection statement—selects between two different actions (or groups of actions).

`switch` statement
Performs one of several actions, based on the value of an expression.
Multiple-selection statement—selects among many different actions (or groups of actions).

Switch Statement activity diagram

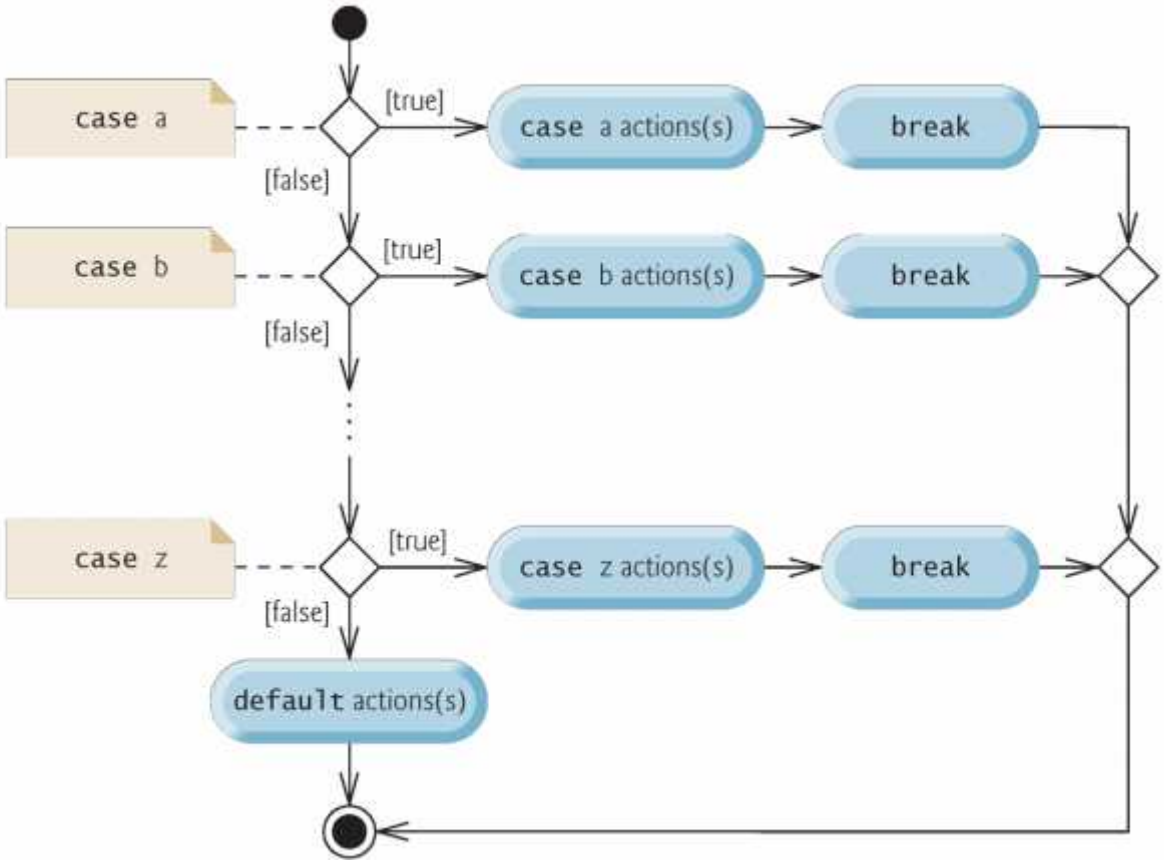


Fig. 4.10 | `switch` multiple-selection statement UML activity diagram with `break` statements.

Nested If Statements

Question: What about a case when there are more than two possible scenarios to make a selection from and its not possible to use Switch statement?

What if the case was that of an age category with three possibilities:

Age > 17,
Age > 17 && Age <35
Age > 35

Use three if structures???

```
if      (Age <= 17)
    System.out.print( “You can’t vote”);

if      (Age > 17  &&  Age <= 35)
     System.out.print( “You can vote, High car Insurance”);

if      (Age > 35)
     System.out.print ( “You can vote, Low car Insurance”);
```

Use nested if Statements

You can test multiple cases by placing `if...else` statements inside other `if...else` statements to create nested *`if...else` statements*.

Using Nested If statements

e.g. 3 case secenario

```
if    (Age > 35)
        System.out.print("Range over  35  ");
else
    if (Age >17)
        System.out.print("Range 18 to 35");
    else
        System.out.print("Range 1 to 18");
```

- Note: Keep all levels of indentation accurate.

.....A note on Errors

Syntax errors (e.g., when one brace in a block is left out of the program) are caught by the compiler.

A logic error (e.g., when both braces in a block are left out of the program) has its effect at execution time.

A fatal logic error causes a program to fail and terminate prematurely.

A nonfatal logic error allows a program to continue executing but causes it to produce incorrect results.

Abbreviated Assignment Expressions

Java provides several assignment operators for abbreviating assignment expressions.

```
c = c + 3;
```

Can be abbreviated with the addition assignment operator += as

```
c += 3;
```

The schema for this is:

```
variable operator = expression
```

Increment and Decrement Operators

Increment and Decrement Operators

Java provides the unary increment operator

++

and the unary decrement operator

--

Increment and Decrement Operators

- If the unary increment or decrement operator is placed after the variable it is referred to as the post increment or post-decrement operator respectively.

- If the unary increment or decrement operator is placed before the variable It is referred to as the pre increment or pre-decrement operator respectively.

It

Therefore ...

Increment and Decrement Operators

- Pre-incrementing (pre -decrementing) a variable causes the variable to be incremented (decremented) by 1,
  then the new value of the variable is used in the expression in which it appears .

- Post-incrementing (post -decrementing) the variable causes the curre nt value of the expression to be used in the expression in which it appears,
  then the variable is incremented (decremented) by 1.

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Arithmetic assignment operators.

Operator	Called	Sample expression	Explanation
`++`	preincrement	`++a`	Increment **a** by 1 then use the new value of **a** in the expression in which **a** resides.
`++`	postincrement	`a++`	Use the current value of **a** in the expression in which **a** resides, then increment **a** by 1.
`--`	predecrement	`--b`	Decrement **b** by 1 then use the new value of **b** in the expression in which **b** resides.
`--`	postdecrement	`b--`	Use the current value of **b** in the expression in which **b** resides, then decrement **b** by 1.

The increment and decrement operators.

Program Example

```
// Fig. 4.14  Increment.java
// Preincrementing and postincrementing

public class Increment {
   public static void main( String args[] )
   {
      int c;

      c = 5;
      System.out.println( c );    // print 5
      System.out.println( c++ ); // print 5 then postincrement
      System.out.println( c );    // print 6

      System.out.println();       // skip a line

      c = 5;
      System.out.println( c );    // print 5
      System.out.println( ++c ); // preincrement then print 6
      System.out.println( c );    // print 6
   }
}
```

```
5
5
6

5
6
6
```

Iteration

Iteration/Repetition

- Many tasks accomplished by repeating some task over and over.
- A brick layer continuously lays block upon block until the wall been constructed is complete.
- Some programming problems are similar.

For Example

- You are so tired after a long days works that you want to write the message

  "***Programming is hard work"***

- 10 times to the screen.

Iteration/Reception

One Solution (Using Sequence)

```
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
System.out.println(“Programming is hard work);
```

Better Solution!

Use the For Statement - Iteration

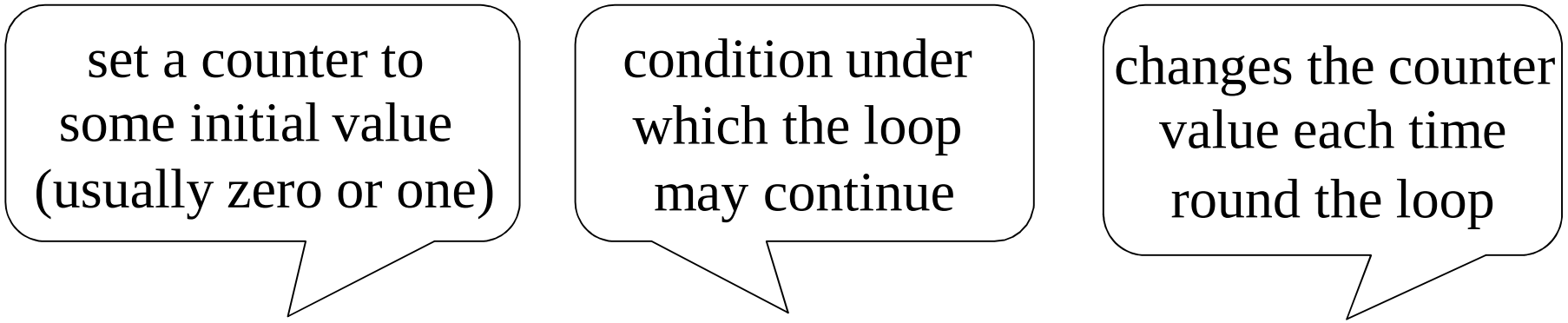
Iteration allows a section of code to be repeated over and over again.

The programming structure that is used to control this repetition is often called a **loop**.

There are three **types of loops** in Java:

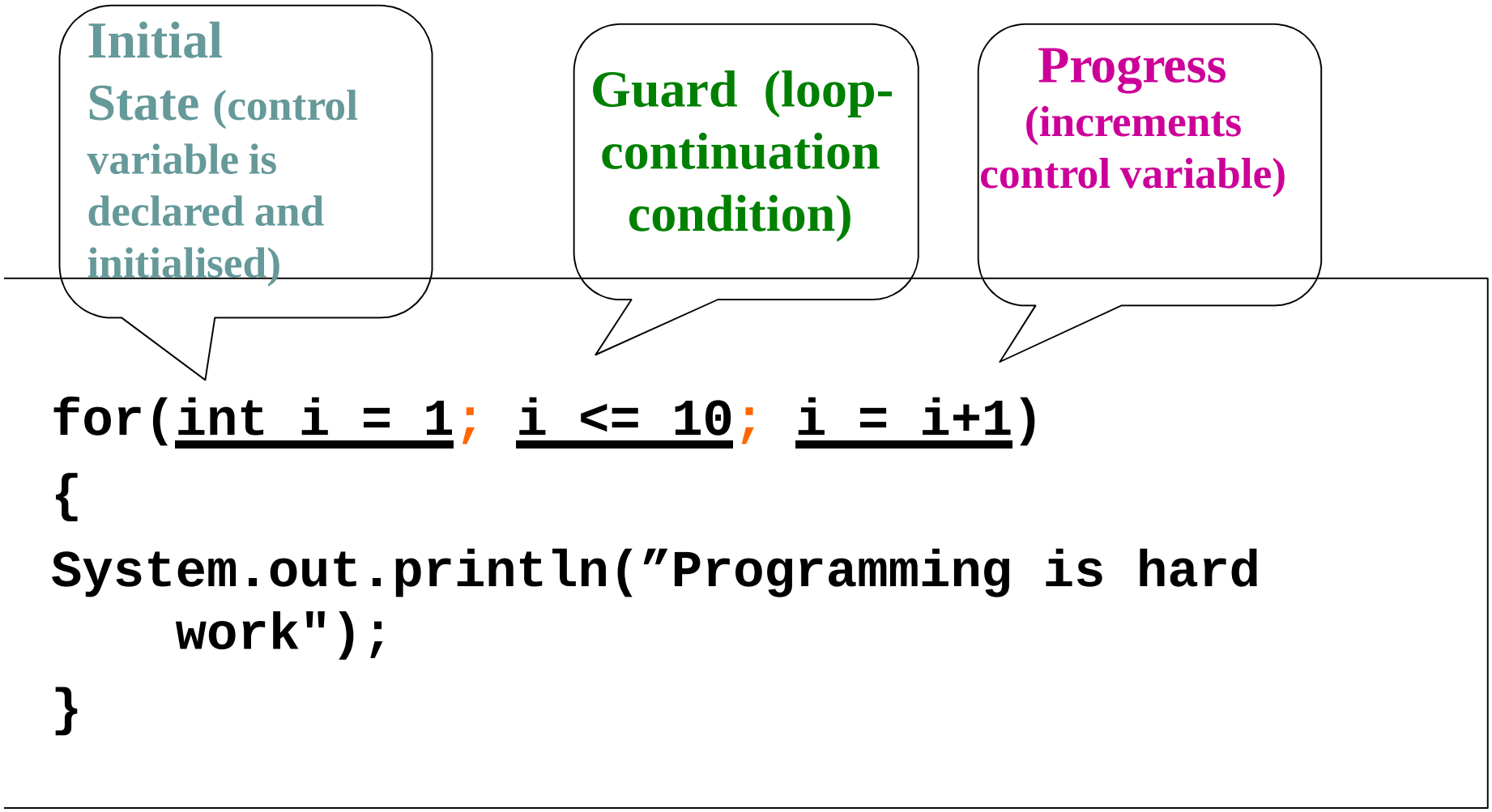
- **for** loop;
- **while** loop;
- **do…while** loop.

The ‘for’ loop




```
for(/*initial state*/;/*guard*/;/*progress*/)
{
      // instruction(s) to be repeated go here
}
```

The ‘for’ loop: an example




```
for(int i = 1; i <= 10; i = i+1)
{
System.out.println(”Programming is hard
    work");
}
```

Semantics of the For Statement

1. Execute the **initial state**
2. Evaluate **guard**
3. Execute task to perform
4. Execute **progress**
5. Repeat steps 2…4 **while guard true**
6. When guard false execute any statements following }, if any

The ‘for’ loop: print numbers 1-10

```
1  // Fig. 4.2: ForCounter.java
2  // Counter-controlled repetition with the for repetition statement.
3
4  public class ForCounter
5  {
6     public static void main( String[] args )
7     {
8        // for statement header includes initialization,
9        // loop-continuation condition and increment
10       for ( int counter = 1; counter <= 10; counter++ )
11          System.out.printf( "%d  ", counter );
12
13       System.out.println(); // output a newline
14    } // end main
15 } // end class ForCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

Fig. 4.2 | Counter-controlled repetition with the `for` repetition statement.

The ‘for’ loop: print numbers 1-10

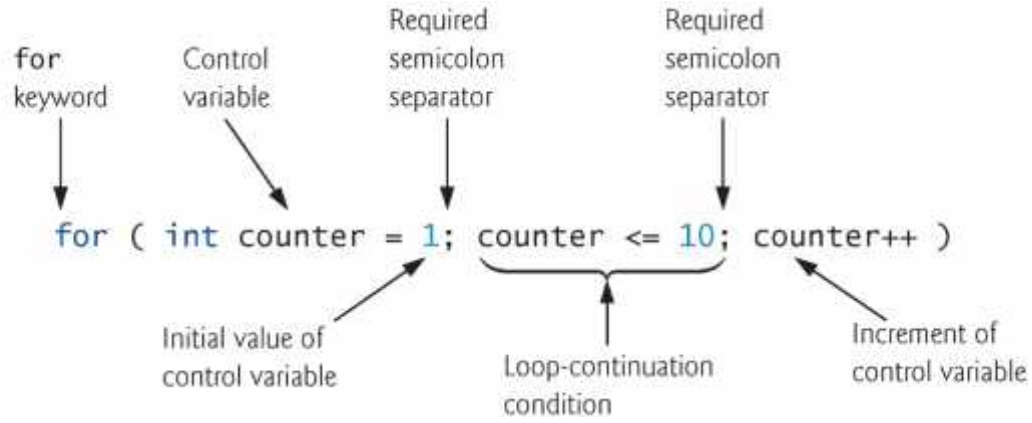


Fig. 4.3 | `for` statement header components.

The ‘for’ loop: UML activity diagram

- The UML activity diagram for a For loop represents both the merge symbol and the decision symbol as diamonds.
- The merge symbol joins two flows of activity into one.
- The decision and merge symbols can be distinguished by the number of “incoming” and “outgoing” transition arrows.
  - A decision symbol has one transition arrow pointing to the diamond and two or more pointing out from it to indicate possible transitions from that point. Each transition arrow pointing out of a decision symbol has a guard condition next to it.
  - A merge symbol has two or more transition arrows pointing to the diamond and only one pointing from the diamond, to indicate multiple activity flows merging to continue the activity. None of the transition arrows associated with a merge symbol has a guard condition.

The ‘for’ loop: print numbers 1-10

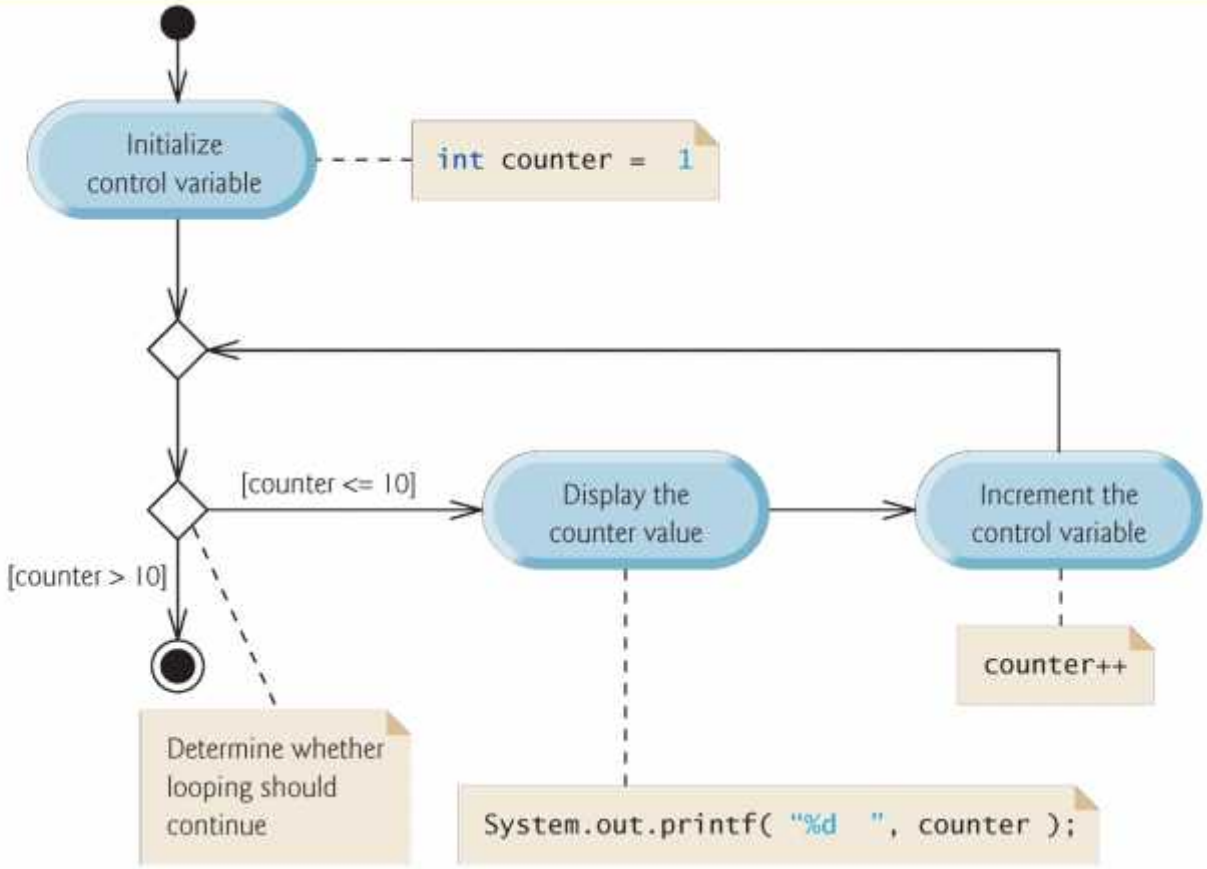


Fig. 4.4 | UML activity diagram for the `for` statement in Fig. 4.2.

For Statement

Examples of
use

Problem:

· Write a piece of code that prints all numbers from 1 to 100.

Solution:

```
for ( int myvar = 1; myvar <= 100; myvar++ )

{
    System.out.println(“Number is “ + myvar);
}
```

For Statement

Problem:

· Write a piece of code that prints all numbers from 100 to 1 in increments of –1, i.e. decrement by 1.

Solution:

```
for ( int myvar = 100; myvar >= 1; myvar  -- )
{
   System.out.println("Number is "  + myvar);
}
```

For Statement

Problem:

· Vary the control variable from 20 to 2 in steps of -2.

Solution:

```
for ( int myvar = 20; myvar >= 2; myvar  -=2 )
```

Problem Example

Write a pseudocode to sum all the integers from 2 to 100.

Pseudocode Solution

```
Set sum to zero
Set number to 2
BeginLoop
        If (number <=100)
           Set sum = sum + number
           Increment number by 2
        EndIf
EndLoop
Print out value of sum
```

Problem Example

Write a program to sum all the integers from 2 to 100.

```
// Summing integers from 2 to 100

public class Sum
{
   public static void main( String args[] )
   {
      int sum = 0;

      for ( int number = 2; number <= 100; number += 2 )
        {
         sum += number;
        }

      System.out.println ("The sum of the numbers from 1 to 100 is " + sum);


      System.exit( 0 );   // terminate the application
   }
}
```

Result

Math.pow

`Math.pow( x, y)` calculates the value of x raised to the power of y.

i.e. x^y

Method `Math.pow( x, y)` takes two arguments of type `double` and returns a double value.

Type `double` is a floating point type (like **float**) but which can store a value of much greater magnitude, and therefore greater precision than **float**.

Math.pow

- Classes provide methods that perform common tasks on objects.
- Java does not include an exponentiation operator—`Math` class `static` method `pow` can be used for raising a value to a power.
- You can call a `static` method by specifying the class name followed by a dot (`.`) and the method name, as in
    - `ClassName.methodName( arguments)`
- `Math.pow(`*x*`,` *y*`)` calculates the value of x raised to the y^{th} power. The method receives two `double` arguments and returns a `double` value.

Problem Example – Math.pow

```
// Using Math.pow to display cube of 2

public class Cube
{
   public static void main( String args[] )
   {
      double sum;

      sum = Math.pow(2,3);


      System.out.println ("The cube of 2 is " + sum);


      System.exit( 0 );   // terminate the application
   }
}
```

Result

The ‘while’ Loop

- Power of computers comes from ability to ask them to do repetitive tasks - iteration very important
- The **for loop** is an often used construct to implement fixed repetitions.
- Sometimes a repetition is required that is not fixed.
- Consider the following scenarios:
  - a racing game that repeatedly moves a car around the track until the car crashes.
  - a password checking program that does not let a user into an application until He/She enters the correct password.

The ‘while’ Loop

- Each of the previous cases involves repetition.
- The number of repetitions is not fixed but depends on some condition.

The `while loop` offers one type of *non-fixed* iteration.

```
while ( /* test goes here */ )
{
  // instruction(s) to be repeated go here
}
```

The ‘while’ Loop

- No need to create a counter to keep track of number of repetitions.

When might this kind of loop be useful??

The ‘while’ Repitition statement Example 1

- Find the first power of 3 larger than 100. Assume `int` variable `product` is initialized to `3`.

```
product = 3;
while ( product <= 100 )
      product = 3 * product;
```

- Each iteration multiplies `product` by 3, so `product` takes on the values 9, 27, 81 and 243 successively.
- When variable `product` becomes 243, the `while`-statement condition—`product <= 100`—becomes false.
- Repetition terminates. The final value of `product` is 243.
- Program execution continues with the next statement after the `while` statement.

The ‘while’ Loop Example 2

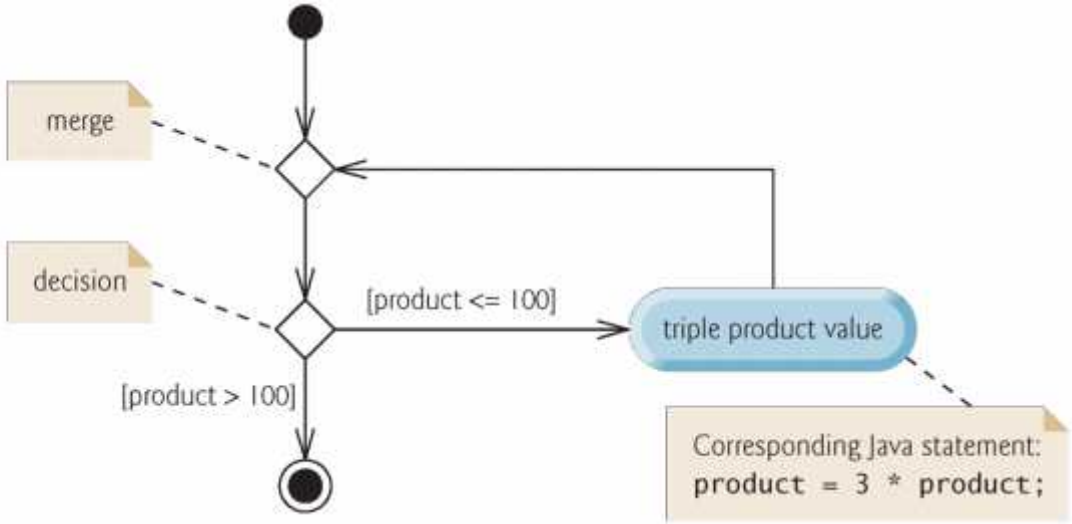


Fig. 3.4 | `while` repetition statement UML activity diagram.

Let's return to a program example from Lecture 3

```
import java.util.Scanner;
class MarkCheck
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int mark;

        System.out.println("What exam mark did you get?");
        mark = input.nextInt();

         if (mark > 13)
         {
           System.out.print("Congratulations you passed!");
         }
         else
         {
            System.out.print("Im sorry but you failed!");

         }

         System.out.println("Good Luck with your other exams");

    }
}
```

The ‘while’ Loop

- A second example is the use of the while loop to **check data** that is input by users.
- Checking input data for errors is referred to as **input validation.**
- In the previous program , the mark that is entered should never be > 100 as test are graded out of 100.
- At the time we **assumed** user would enter mark correctly.
- **Good programmers never make this assumption**.

- Good practice to check that the **mark** entered is **valid.**
- If not the user will be allowed to enter the mark again **until** a **valid mark is entered**.

The ‘while’ Loop

Can express this as **pseudocode** as follows:

```
DISPLAY prompt for mark
ENTER mark
KEEP REPEATING WHILE mark typed in > 100
BEGIN
        DISPLAY error message to user
        ENTER mark
END
// Rest of program here
```

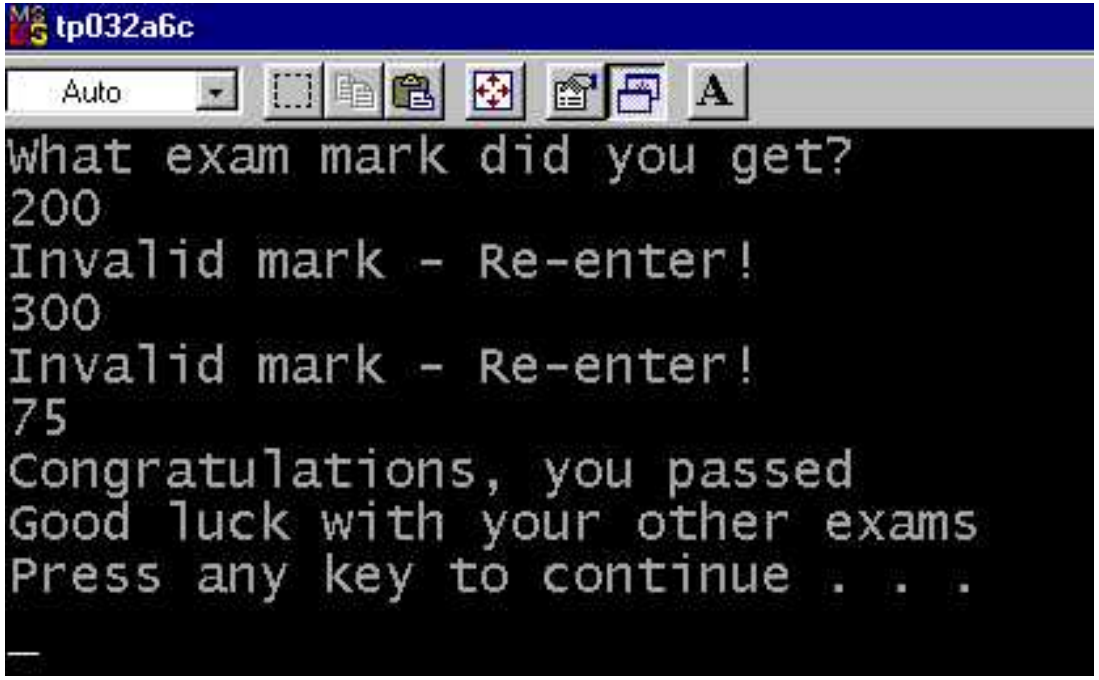
The ‘while’ Loop

Program Code is as follows:

```
import java.util.Scanner;
class MarkCheck
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int mark;

        System.out.println("What exam mark did you get?");
        mark = input.nextInt();

        while (mark > 100)
        {
            System.out.println("invalid mark!! - Re-enter!");
            mark = input.nextInt();
        }


         if (mark > 39)
         {
           System.out.print("Congratulations you passed!");
         }
         else
         {
            System.out.print("Im sorry but you failed!");

         }

         System.out.println("Good Luck with your other exams");

    }
}
```

The ‘while’ Loop

Example of Program Result



Using Logical Operators

- The mark should also **never be less than zero**.
- This requires a more **complicated test** condition.
- We want our test to say **(mark>100 OR mark<0)**
- Word ‘OR’ not a valid Java word
- Have to use the following symbol **||**
- Symbols like OR and AND are known as logical or boolean operators.

Using Logical Operators

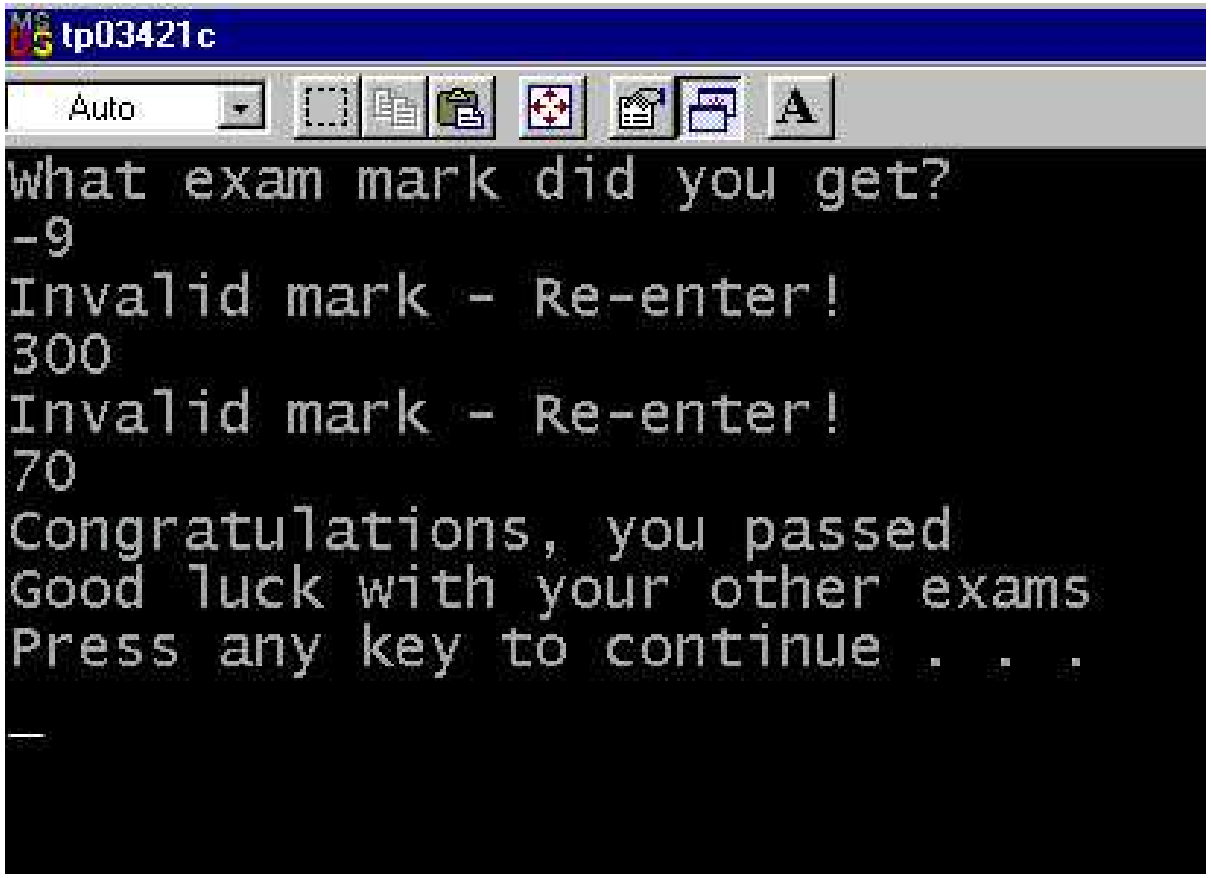
The logical operators of Java	
Logical operator	**Java counterpart**
AND	&&
OR	||
NOT	!

```
while (mark < 0 || mark > 100)
{
  // instruction(s) to be repeated go here
}
```

Program Example

```
import java.util.Scanner;
class MarkCheckWhileOr
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int mark;

        System.out.println("What exam mark did you get?");
        mark = input.nextInt();

        while (mark < 0 || mark > 100)
        {
            System.out.println("invalid mark!! - Re-enter!");
            mark = input.nextInt();
        }


         if (mark > 39)
         {
           System.out.print("Congratulations you passed!");
         }
         else
         {
            System.out.print("Im sorry but you failed!");

         }

         System.out.println("Good Luck with your other exams");

    }
}
```

Sample Test Run



Example Logical And - &&:

```
If ( gender == 1 && age >= 65 )
    ++seniorFemales;
```

```
If ( gender == 1 ) && ( age >= 65 )
    ++seniorFemales;
```

· The above constructs are equivalent

This condition is true if and only if both of the simple conditions are true.

That is, if this combined condition is true then the count of seniorFemales is incremented.

Example Logical OR ||

```
If ( semesterAverage >= 90  || final Exam >= 90  )
     System.out.println(  “Student grade is A” );
```

This statement contains two simple conditions.

The if statement considers the combined condition and awards the student an “A” if either or both of the simple conditions are true.

Counter Controlled Repetition

Formulating Algorithms:
Counter-Controlled Repitition

- `A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.`
- The class average is equal to the sum of the grades divided by the number of students.
- The algorithm for solving this problem on a computer must input each grade, keep track of the total of all grades input, perform the averaging calculation and print the result.
- Use counter-controlled repetition to input the grades one at a time.
- A variable called a counter (or control variable) controls the number of times a set of statements will execute.
- Counter-controlled repetition is often called definite repetition, because the number of repetitions is known before the loop begins executing.

Formulating Algorithms:
Counter-Controlled Repitition

- A total is a variable used to accumulate the sum of several values.
- A counter is a variable used to count.
- Variables used to store totals are normally initialized to zero before being used in a program.

Pseudocode

```
1  Set total to zero
2  Set grade counter to one
3
4  While grade counter is less than or equal to ten
5      Prompt the user to enter the next grade
6      Input the next grade
7      Add the grade into the total
8      Add one to the grade counter
9
10 Set the class average to the total divided by ten
11 Print the class average
```

Java Code

```
1  // Fig. 3.6: ClassAverage.java
2  // Counter-controlled repetition: Class-average problem.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class ClassAverage
6  {
7     public static void main( String[] args )
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner( System.in );
11
12       int total; // sum of grades entered by user
13       int gradeCounter; // number of the grade to be entered next
14       int grade; // grade value entered by user
15       int average; // average of grades
16
17       // initialization phase
18       total = 0; // initialize total
19       gradeCounter = 1; // initialize loop counter
20
```

Java Code

```
21          // processing phase
22          while ( gradeCounter <= 10 ) // loop 10 times
23          {
24             System.out.print( "Enter grade: " ); // prompt
25             grade = input.nextInt(); // input next grade
26             total = total + grade; // add grade to total
27             gradeCounter = gradeCounter + 1; // increment counter by 1
28          } // end while
29
30          // termination phase
31          average = total / 10; // integer division yields integer result
32
33          // display total and average of grades
34          System.out.printf( "\nTotal of all 10 grades is %d\n", total );
35          System.out.printf( "Class average is %d\n", average );
36       } // end main
37    } // end class ClassAverage
```

On execution….

```
Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Program note on execution….

- The program's output indicates that the sum of the grade values in the sample execution is 846, which, when divided by 10, should yield the floating-point number 84.6.
- The result of the calculation `total / 10` is the integer 84, because `total` and `10` are both integers.
- Dividing two integers results in integer division—any fractional part of the calculation is lost (i.e., truncated).

Sentinel-Controlled Repetition

Formulating Algorithms: Sentinel-Controlled Repitition

Let's generalise the class average problem. Consider the following problem:

Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

- In the previous example, the number of grades(10) was **known** in advance
- In this example **no indication** of how many grades are to be entered.
- The program must process an **arbitrary** number of grades.

How can the program determine when to stop input of grades?

Use a special value called a sentinel value

How does the sentinel value work??

- The user types in grades until all grades have been entered.
- The user types in the sentinel value (usually -1) to indicate that the last grade has been entered.

Pseudocode

```
Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)
While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
else
    Print "No grades were entered"
```

Fig. 4.8 Pseudocode algorithm that uses sentinel-controlled repetition to solve the class-average problem

Program Code

```
1  // Fig. 3.8: ClassAverage.java
2  // Sentinel-controlled repetition: Class-average problem.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class ClassAverage
6  {
7     public static void main( String[] args )
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner( System.in );
11
12       int total; // sum of grades
13       int gradeCounter; // number of grades entered
14       int grade; // grade value
15       double average; // number with decimal point for average
16
17       // initialization phase
18       total = 0; // initialize total
19       gradeCounter = 0; // initialize loop counter
20
21       // processing phase
22       // prompt for input and read grade from user
23       System.out.print( "Enter grade or -1 to quit: " );
24       grade = input.nextInt();
```

Program Code

```
25
26          // loop until sentinel value read from user
27          while ( grade != -1 )
28          {
29             total = total + grade; // add grade to total
30             gradeCounter = gradeCounter + 1; // increment counter
31
32             // prompt for input and read next grade from user
33             System.out.print( "Enter grade or -1 to quit: " );
34             grade = input.nextInt();
35          } // end while
36
37          // termination phase
38          // if user entered at least one grade...
39          if ( gradeCounter != 0 )
40          {
41             // calculate average of all grades entered
42             average = (double) total / gradeCounter;
43
44             // display total and average (with two digits of precision)
45             System.out.printf( "\nTotal of the %d grades entered is %d\n",
46                gradeCounter, total );
47             System.out.printf( "Class average is %.2f\n", average );
48          } // end if
```

Program Code

```
49          else // no grades were entered, so output appropriate message
50             System.out.println( "No grades were entered" );
51       } // end main
52    } // end class ClassAverage
```

```
Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

Program Logic

- Program logic for sentinel-controlled repetition
  - Reads the first value before reaching the `while`.
  - This value determines whether the program's flow of control should enter the body of the `while`. If the condition of the `while` is false, the user entered the sentinel value, so the body of the `while` does not execute (i.e., no grades were entered).
  - If the condition is true, the body begins execution and processes the input.
  - Then the loop body inputs the next value from the user before the end of the loop.

Program Notes

- Averages do **not always** evaluate to integer values. Often an average is a value such as 3.333 or 2.7 that contains a fractional part.
- These values are referred to as floating point numbers and are represented by data type **double**. Variable
- average is declared as type double. However the result of total/gradeCounter is an **integer** because total and gradeCounter are integers variables.
- To produce a floating-point calculation with integer values, we must create temporary values that are floating-point numbers for the calculation.

Java provides the unary cast operator to accomplish this task

Program Notes

Line 39

average = (double) total/gradeCounter

uses the cast operator (double) to create a temporary floating-point copy of its operand **total.** As a result the value stored in average will be a floating-point number.

Formulating Algorithms: Nested Control Statements

- This case study examines nesting one control statement within another.
- A college offers a course that prepares students for the state licensing exam for real estate brokers. Last year, ten of the students who completed this course took the exam. The college wants to know how well its students did on the exam. You've been asked to write a program to summarize the results. You've been given a list of these 10 students. Next to each name is written a 1 if the student passed the exam or a 2 if the student failed.

Formulating Algorithms: Nested Control Statements

- This case study examines nesting one control statement within another.
- Your program should analyze the results of the exam as follows:
  - Input each test result (i.e., a 1 or a 2). Display the message "Enter result" on the screen each time the program requests another test result.
  - Count the number of test results of each type.
  - Display a summary of the test results, indicating the number of students who passed and the number who failed.
  - If more than eight students passed the exam, print the message "Bonus to instructor!"

Pseudocode

```
1   Initialize passes to zero
2   Initialize failures to zero
3   Initialize student counter to one
4
5   While student counter is less than or equal to 10
6       Prompt the user to enter the next exam result
7       Input the next exam result
8
9       If the student passed
10          Add one to passes
11      Else
12          Add one to failures
13
14      Add one to student counter
15
16  Print the number of passes
17  Print the number of failures
18
19  If more than eight students passed
20      Print "Bonus to instructor!"
```

```
1  // Fig. 3.10: Analysis.java
2  // Analysis of examination results.
3  import java.util.Scanner; // class uses class Scanner
4
5  public class Analysis
6  {
7     public static void main( String[] args )
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner( System.in );
11
12       // initializing variables in declarations
13       int passes = 0; // number of passes
14       int failures = 0; // number of failures
15       int studentCounter = 1; // student counter
16       int result; // one exam result (obtains value from user)
17
18       // process 10 students using counter-controlled loop
19       while ( studentCounter <= 10 )
20       {
21          // prompt user for input and obtain value from user
22          System.out.print( "Enter result (1 = pass, 2 = fail): " );
23          result = input.nextInt();
```

Fig. 3.10 | Nested control structures: Examination-results problem. (Part 1 of 4.)

```
24
25             // if...else nested in while
26             if ( result == 1 )           // if result 1,
27                passes = passes + 1;      // increment passes;
28             else                         // else result is not 1, so
29                failures = failures + 1; // increment failures
30
31             // increment studentCounter so loop eventually terminates
32             studentCounter = studentCounter + 1;
33          } // end while
34
35          // termination phase; prepare and display results
36          System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
37
38          // determine whether more than 8 students passed
39          if ( passes > 8 )
40             System.out.println( "Bonus to instructor!" );
41       } // end main
42    } // end class Analysis
```

Fig. 3.10 | Nested control structures: Examination-results problem. (Part 2 of 4.)

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

Fig. 3.10 | Nested control structures: Examination-results problem. (Part 3 of 4.)

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

Fig. 3.10 | Nested control structures: Examination-results problem. (Part 4 of 4.)

Do/While Structure

The do/while Repetition Structure

The do/while repetition structure is similar to the while structure.

BUT

...

In the while structure the loop condition is tested at the beginning of the loop before the body of the loop is performed.

The do/while structure tests the loop-continuation condition after the body of the loop is performed.

In the do/while structure the body of the loop is always executed at least once.

The do/while Repetition Structure

```
1  // Fig. 4.7: DoWhileTest.java
2  // do...while repetition statement.
3
4  public class DoWhileTest
5  {
6     public static void main( String[] args )
7     {
8        int counter = 1; // initialize counter
9
10       do
11       {
12          System.out.printf( "%d  ", counter );
13          ++counter;
14       } while ( counter <= 10 ); // end do...while
15
16       System.out.println(); // outputs a newline
17    } // end main
18 } // end class DoWhileTest
```

```
1  2  3  4  5  6  7  8  9  10
```

Fig. 4.7 | do...while repetition statement.

The do/while Repetition Structure

- The following slide contains the UML activity diagram for the `do…while` statement.
- The diagram makes it clear that the loop-continuation condition is not evaluated until after the loop performs the action state at least once.

The do/while Repetition Structure

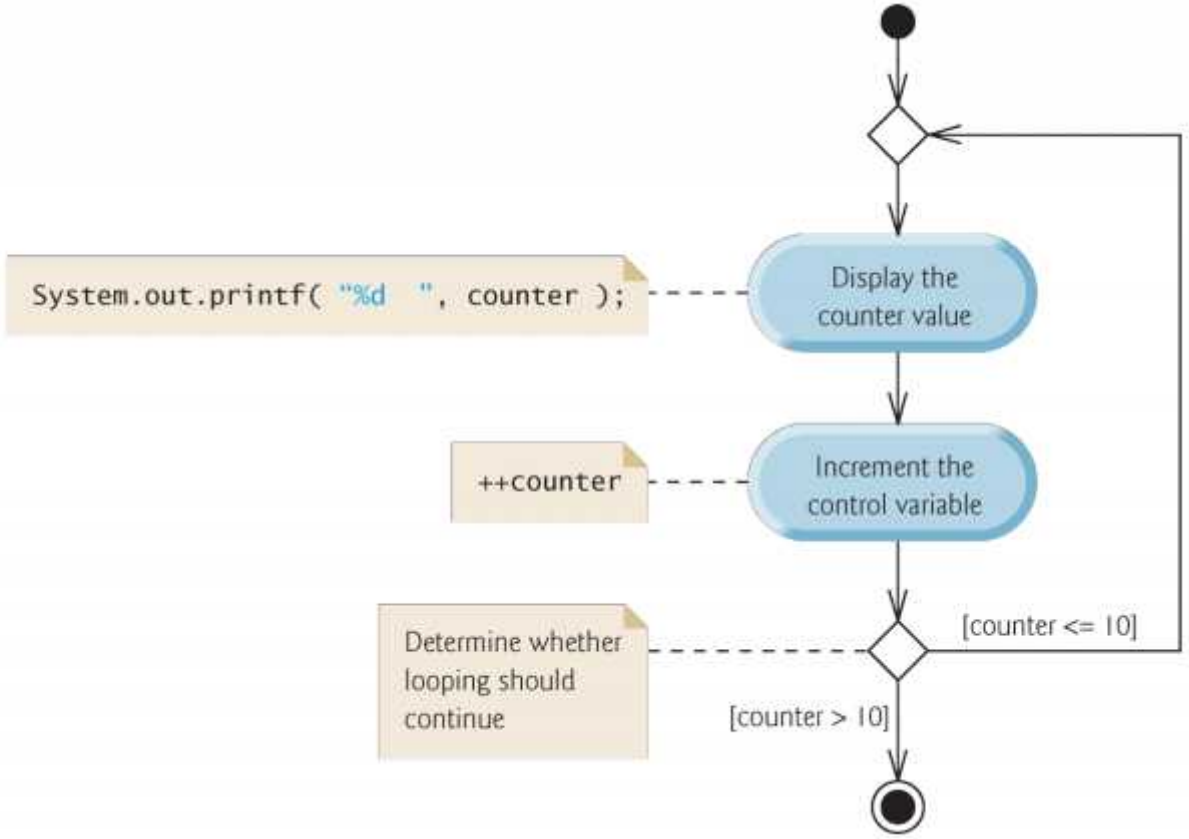


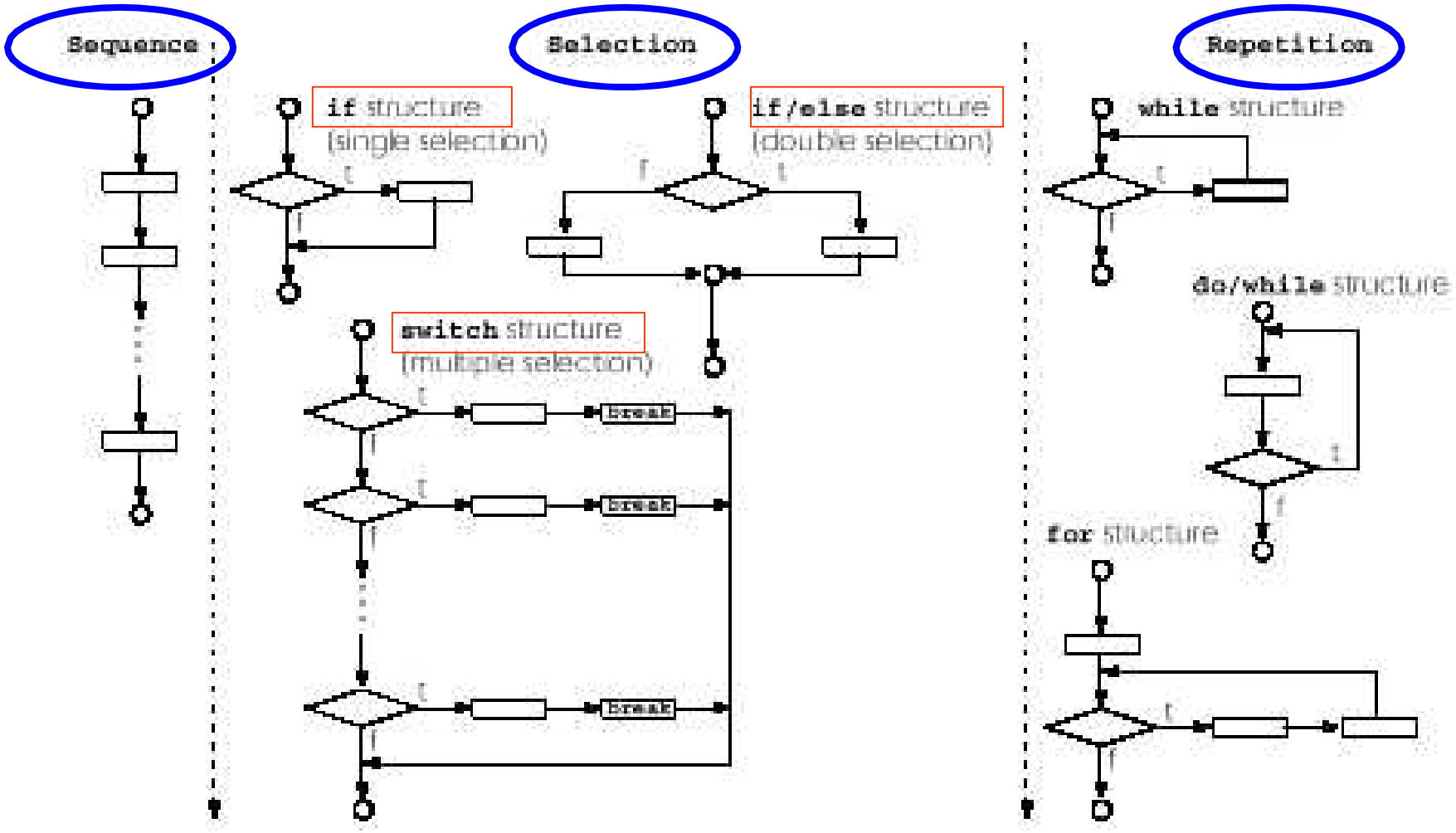
Fig. 4.8 | `do...while` repetition statement UML activity diagram.

An example of a menu driven program

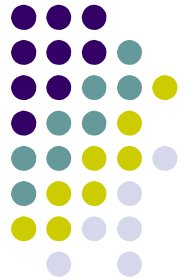
```
int response;
System.out.println("***Lab Times***");
do
{
   System.out.println(); System.out.println("[1]
   TIME FOR GROUP A"); System.out.println("[2]
   TIME FOR GROUP B"); System.out.println("[3]
   TIME FOR GROUP C"); System.out.println("[4]
   QUIT PROGRAM"); System.out.print("enter
   choice [1,2,3,4]: "); response =
   input.nextInt(); System.out.println();
   switch(response)
   {
     case 1: System.out.println("10.00 a.m ");break;
     case 2: System.out.println("1.00 p.m ");break;
     case 3: System.out.println("11.00 a.m ");break;
     case 4: System.out.println("Goodbye ");break;
     default:  System.out.println("Options 1-4 only!");
   }
} while (response != 4);
```

Structured Programming Using Control Structures

Summary Diagram– 3 Types of Control Structures



What have we achieved so far



AIMS

1. To introduce you to the fundamental concepts of software development, with an emphasis on problem solving & computer programming.

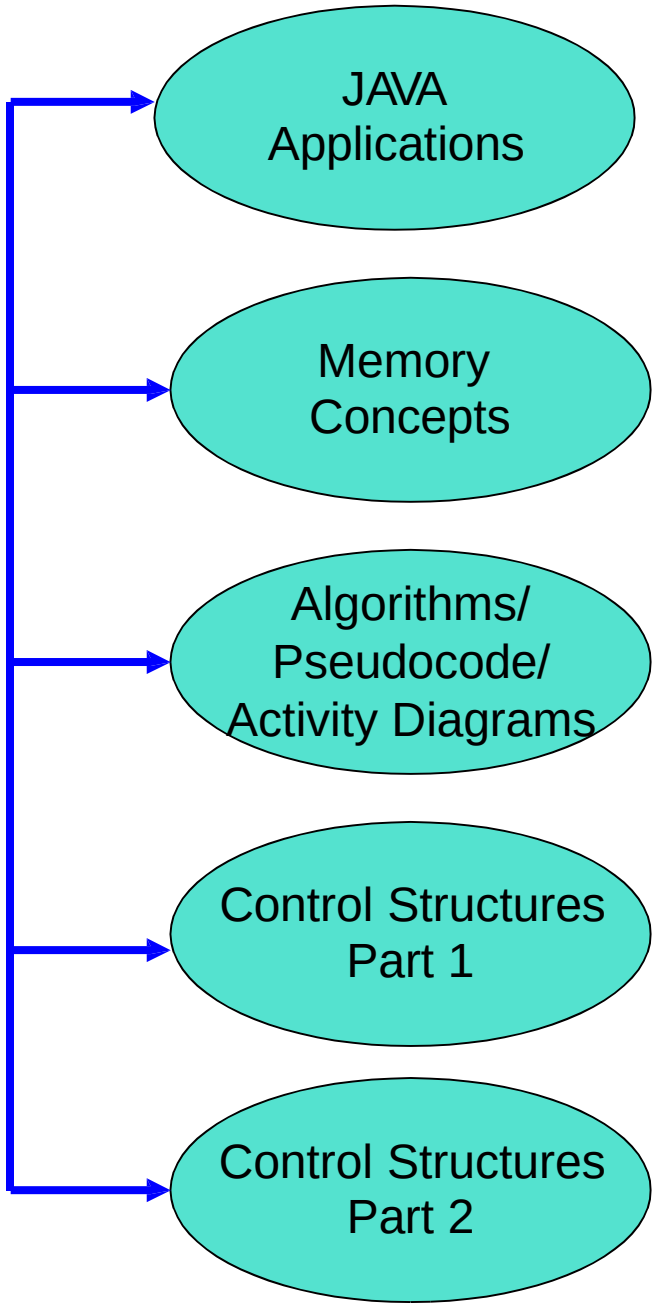
2. To teach you a structured approach to problem solving and computer programming

3. To teach you the fundamentals of programming in Java

and you now have:

- Developed problem solving skills
- A working knowledge of good programming practice
- An ability to write well structured programs
- Have practical experience of designing, coding & de - bugging programs in Java

WHAT WE HAVE COVERED SO FAR



- History of Java
- Java Applications
- Input/Output from Keyboard

- Data Types, Variables
- Arithmetic Operators
- Rules of operator precedence

- Algorithms
- Pseudocode
- Activity Diagrams

- If If/Else
- Switch

- Repetition
- For , While, Do/While
- Counter controlled/ Sentinel Repetition