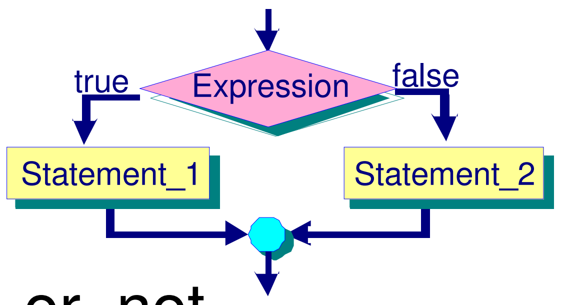
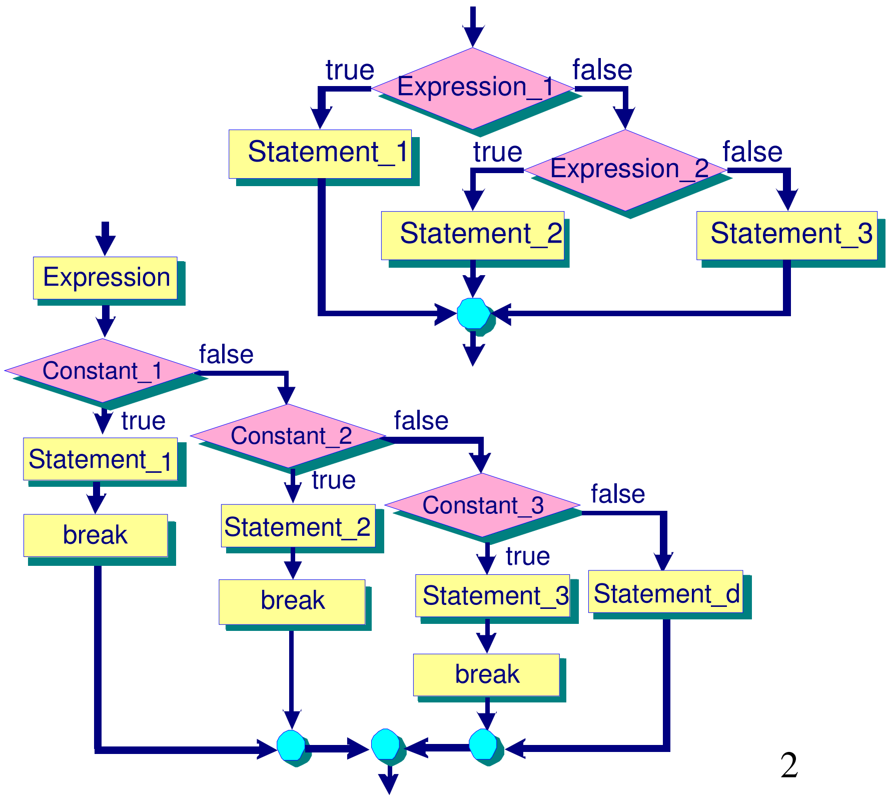
Chapter 7

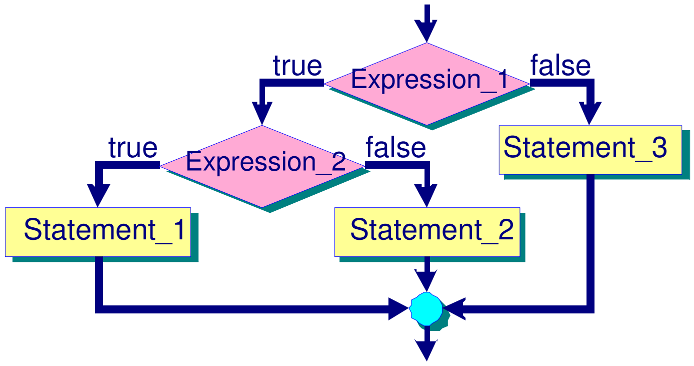
Looping

Review: Branching

- Branching (or Selection)
  - for making decision
- Relational and Logical Operators
  - equal to, greater than, less than; and, or, not
- The if and if-else Statements
- The if-else if-else Statement
- The Nested-if Statement
- The switch Statement
- The Conditional Operator







Looping

- **Why Loops?**
- The while Statement
- The for Statement
- The do while Statement
- The break and continue Statement
- Nested Loops
- Case Study

Key: Understand the logic!!!

Why Loops?

- The branching **if-else** and **switch** statements enable us to make **selection**.

- Sometimes, we need statements to execute actions **repeatedly**.

- Ex 1: consider a multiplication table for a given number:

9 Multiplication table

1 x 9 = 9
2 x 9 = 18
…
9 x 9 = 81
10 x 9 = 90

```
int num;
// read in num from user (e.g. user enters 9)
// print the table
System.out.println(“1 x “ + num + “ = “ + (1 * num));
System.out.println(“2 x “ + num + “ = “ + (2 * num));
…
System.out.println(“9 x “ + num + “ = “ + (9 * num));
System.out.println(“10 x “ + num + “ = “ + (10 * num));
```

REPEATING WITH CERTAIN PATTERNS

- You have **10** sets of println() statements!!!
- Is this necessary??

- Example 2: Find the average mark scored by 50 students in the Java Programming course.

```
int score1, score2, …;
// you need to read in score for each student & add them
score1 = sc.nextDouble();
score2 = sc.nextDouble();
…
score50 = sc.nextDouble();
double total = score1 + … + score50;
// then compute the average
double average = total/50.0;
System.out.println("Average = " + average);
```

REPEATING WITH CERTAIN PATTERNS

- You need to define **50** variables to hold the values of student mark!!!
- Is this necessary??

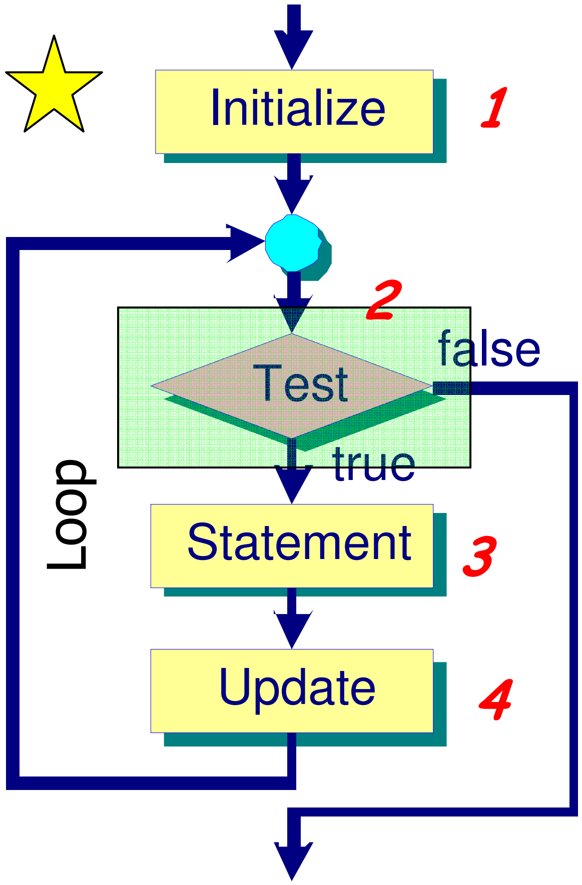
Looping (or Repetition)

To construct loops, we usually need:

1. **Initialize** – initialize the loop control variable.

Loop

2. **Test condition** – evaluate the test condition (involve loop control variable).
3. **Loop body** – the loop body is executed if test is true.
4. **Update** – typically, loop control variable is modified through the execution of the loop body. It can then go through the **test** condition.



➔**Need to setup Loop Control Variable in (1)**

- From **Example 2**: Find the average mark scored by 50 students in the Java Programming course.

Algorithm in Pesudocode:

Understand the logic!!!

```
main:
     SET total TO 0                     // Initialize
1    SET counter TO 0
2    WHILE counter < 50                 // Test
   3      READ mark                     // Loop body
          ADD mark TO total
4         INCREMENT counter BY 1        // Update
     ENDWHILE
     COMPUTE average = total/counter
     PRINT average
```

What/where is the Loop Control Variable ???

Program Execution

Inputs/initialization:

Trace it!!!!!

counter = 0, mark = 0, total = 0

counter	counter < 50	Read mark	Add total	Loop No.	Output average
0	true	55	55	1	
1	true	45	100	2	
…	…	…	…	…	
49	true	55	2500	50	
50	false			exit	2500/50 = 50

```
  SET total TO 0                         // Initialize
1 SET counter TO 0
2 WHILE counter < 50                     // Test
3      READ mark                         // Loop body
       ADD mark TO total
4      INCREMENT counter BY 1    // Update
  ENDWHILE
  COMPUTE average = total/counter
  PRINT average
```

Outputs:

Average = 50

Looping

There are **three types** of looping statements:

- while
- for
- do while

NB: **Not** all of them contain the **4 steps** of looping structure in its declarations explicitly.

Looping

- Why Loops?
- **The while Statement**
- The for Statement
- The do while Statement
- The break and continue Statement
- Nested Loops
- Case Study

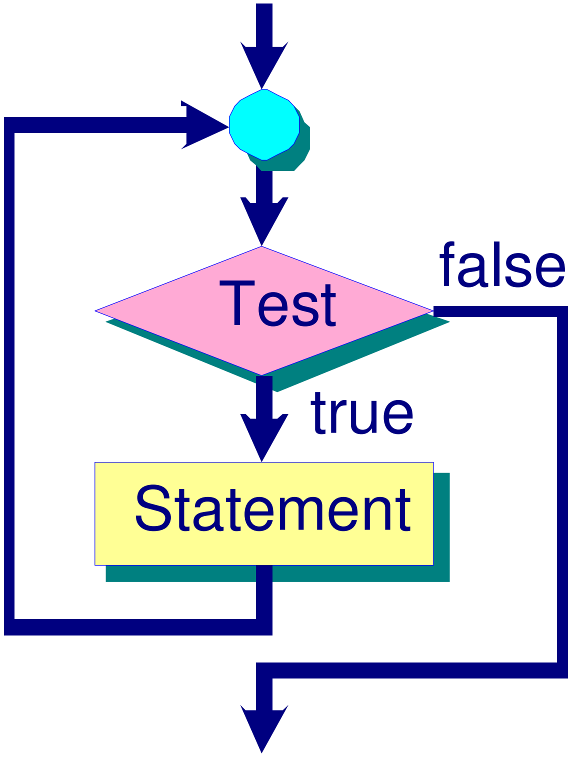
The while Loop

The format of the *while* statement is

```
while (Test)
    Statement
```

Statement can be
(1) a simple statement
    terminated by a semicolon
    or
(2) a compound statement
    enclosed by { }

Logic of while loop



Types of Loops

There are two types of loops:

- **Counter-controlled loops** – the loop body is repeated for a number of times, and the ***number of repetitions is known*** before the loop starts execution.

- **Sentinel-controlled loops** – ***the number of repetitions is NOT known*** before the loop starts execution. Usually, a **sentinel value** (such as –1, different from the regular data) is used to determine whether to execute the loop body.

- **Example: Counter-Controlled Loop**

```
import java.text.*; import java.util.Scanner;
public class ComputeAverage {
  public static void main(String[] args) {
      double total=0.0, mark=0.0;
 1    int counter=0;
      Scanner sc = new Scanner(System.in);

 2    while (counter < 50)
      {
        mark = sc.nextDouble();
   3    total += mark;
 4      counter++;
      }
      average = total/counter;
      System.out.println(“Average + ‟ + average);
  }
}
```

counter - Loop Control Variable

The number of execution is fixed depending on counter

NB: In every loop, there must be a point in the loop body to make the loop condition become false -> otherwise infinite loop

Program Execution

Same as the pseudo code trace

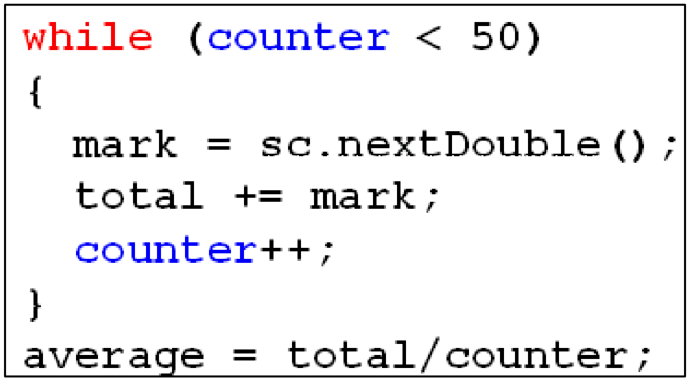
Inputs/initialization:

`counter` = 0, `mark` = 0, `total` = 0

counter	counter < 50	mark	total	Output average
0	true	55	55	
1	true	45	100	
…	…	…	…	
49	true	55	2500	
50	**false**			2500/50 = 50

Outputs:

Average = 50



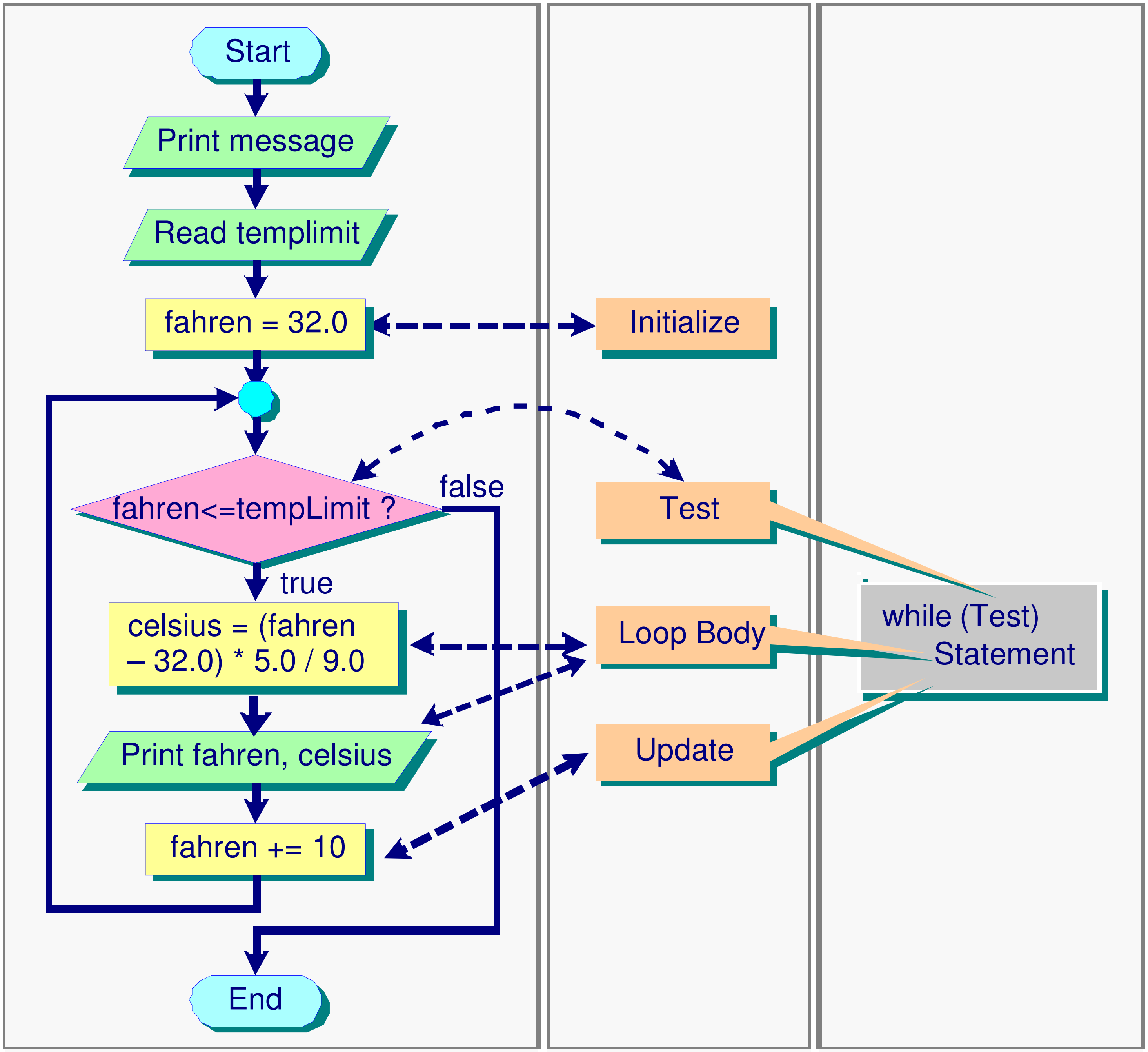
```
while (counter < 50)
{
  mark = sc.nextDouble();
  total += mark;
  counter++;
}
average = total/counter;
```

Example: Counter-Controlled Loop- Computing Temperature

```
import java.text.*; import java.util.Scanner;
public class ConvertTemp {
   public static void main(String[] args) {
      double fahren, celsius;
      double tempLimit;
      Scanner sc = new Scanner(System.in);
      DecimalFormat numForm =
            new DecimalFormat("000.00");
      System.out.println("Enter conversion limit(F: ");
      tempLimit = sc.nextDouble();
      System.out.println("\tFahrenheit\tCelsius");
      System.out.println("\t----------\t-------");
      fahren = 32.0;
      while (fahren <= tempLimit) {
        celsius = (fahren - 32.0) * 5.0/9.0;
        System.out.println("\t " + numForm.format(fahren)
             + "\t\t\t" + numForm.format(celsius));
        fahren += 10;
      }
   }
}
```

The number of execution is fixed depending on tempLimit

fahren - Loop Control Variable

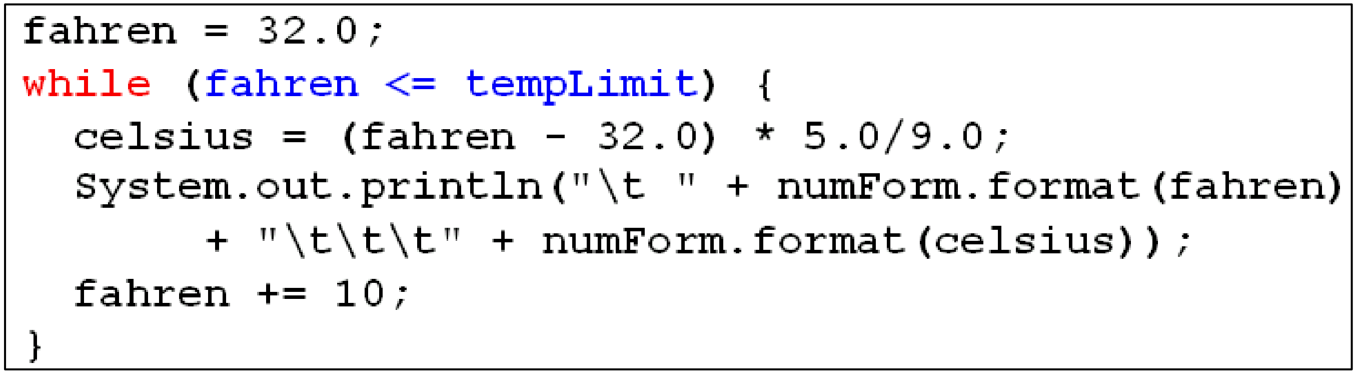

Start
Print message
Read templimit
fahren = 32.0
fahren<=tempLimit ?
false
true
celsius = (fahren – 32.0) * 5.0 / 9.0
Print fahren, celsius
fahren += 10
End
Initialize
Test
Loop Body
Update
while (Test)
Statement

Program Execution

`Inputs/initialization:`

`tempLimit` = 112.0, **`fahren`** = 32.0

`fahren`	**`fahren < tempLimit`**	**`Output`** **`celsius`**
32.0	true	0
42.0	true	5.56
…	…	…
112.0	true	44.44
122.0	**false**	



```
fahren = 32.0;
while (fahren <= tempLimit) {
  celsius = (fahren - 32.0) * 5.0/9.0;
  System.out.println("\t " + numForm.format(fahren)
          + "\t\t\t" + numForm.format(celsius));
  fahren += 10;
}
```

Program Input and Output

```
Enter the conversion limit (F): 112.0
     Fahrenheit     Celsius
     ----------     -------
      032.00         000.00
      042.00         005.56
      052.00         011.11
      062.00         016.67
      072.00         022.22
      082.00         027.78
      092.00         033.33
      102.00         038.89
      112.00         044.44
```

- **Example: Sentinel-Controlled Loop**

```
import java.text.*; import java.util.Scanner;
public class ComputeAverage {
  public static void main(String[] args) {
      double total=0.0;
      int counter=0;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter mark: ");
1     double mark = sc.nextDouble();
2     while (mark != -1){
        total += mark;
3       counter++;
        System.out.println("Enter mark: ");
4       mark = sc.nextDouble();
      }
      if (counter != 0) {
        average = total/counter;
        System.out.println("Average + " + average);
      }
  }
}
```

mark - Loop Control Variable

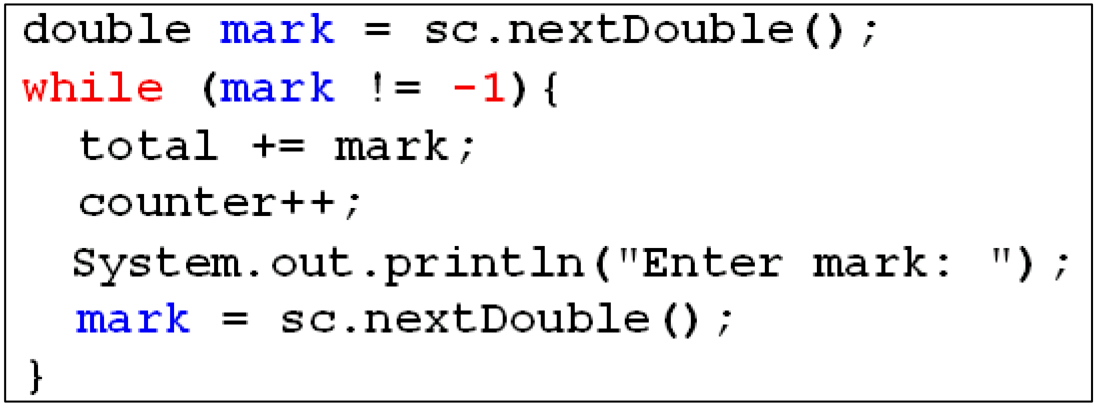
The number of marks to be input depends on the number of students who took the exam, which is unknown

Program Execution

Inputs/initialization:

counter = 0, **mark** = 0, **total** = 0

mark	**mark != -1**	**counter**	**total**	**Output average**
55	true	1	55	
45	true	2	100	
…	…	…	…	
65	true	10	550	
-1	**false**			550/10 = 55



```
double mark = sc.nextDouble();
while (mark != -1){
  total += mark;
  counter++;
  System.out.println("Enter mark: ");
  mark = sc.nextDouble();
}
```

Outputs:

Average = 55

Example: Sentinel-Controlled Loop

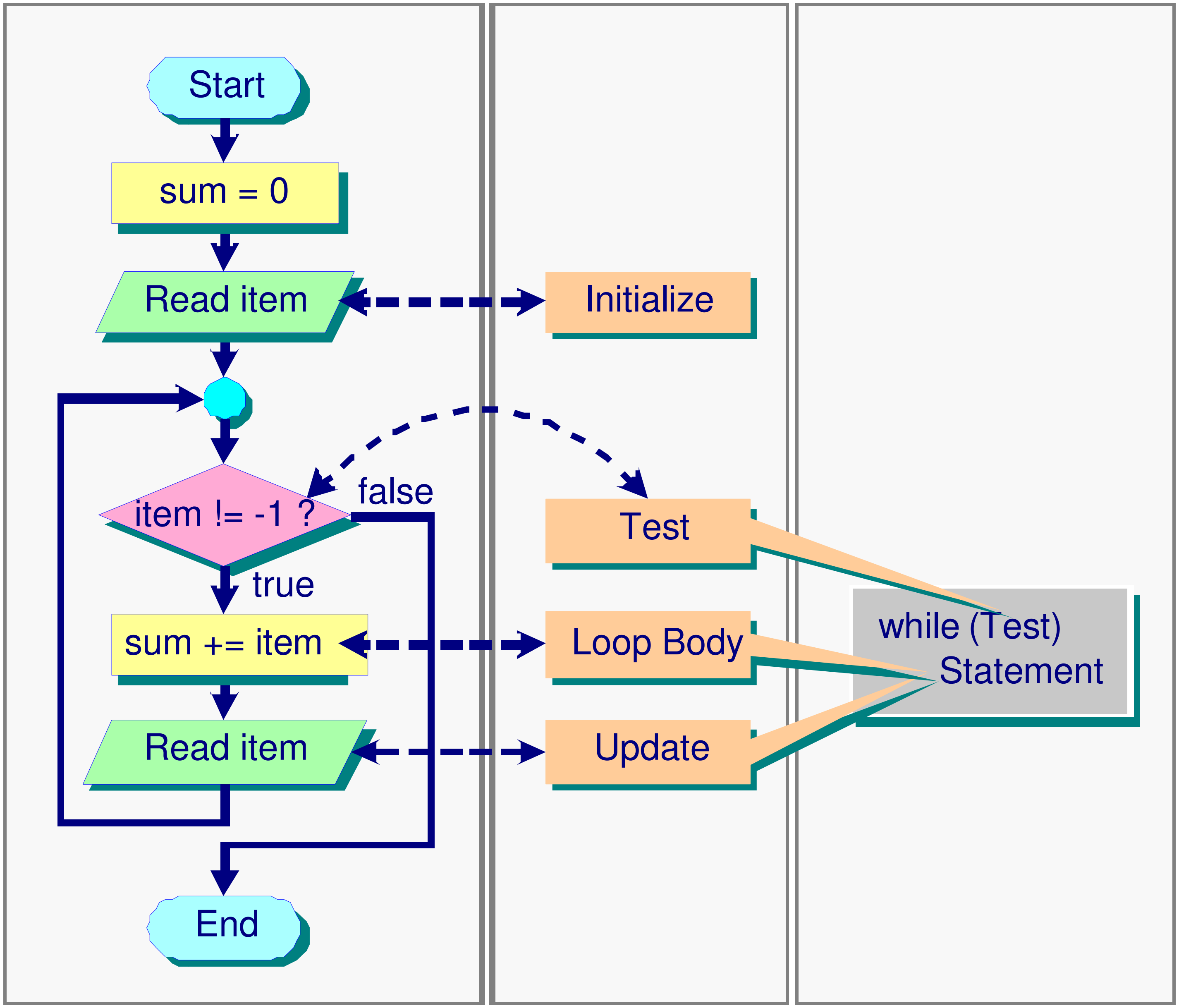
```
Import java.util.Scanner;
public class CalculateSum {
   public static void main(String[] args){
      int sum=0, item;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the list of integers: ");
      item = sc.nextInt();
      while (item != -1) {
         /* sentinel loop controlled loop */
         sum += item;
         item = sc.nextInt();
      }
      System.out.println("The sum is " + sum);
   }
}
```

The number of inputs is unknown

Program Input and Output

```
Enter the list of integers:
1 8 11 24 36 48 67 -1
The sum is 195

Enter the list of integers:
-1
The sum is 0
```


Start
sum = 0
Read item
Initialize
item != -1 ?
false
Test
true
sum += item
Loop Body
while (Test)
Statement
Read item
Update
End

Program Execution

Inputs/initialization:

sum= 0, **item** = 0

item	**item != -1**	**sum**
1	true	1
8	true	9
…	…	…
67	true	195
-1	**false**	

Outputs:

The sum is 195

Example: Finding the power of a number using while loop

```
import java.util.Scanner;
public class PowerApp {                          // compute x^y
   public static void main(String[] args) {
      double x; int y; double result = 1.0;
      Scanner sc = new Scanner(System.in);
      System.out.println("Please enter x: ");
      x = sc.nextDouble();
      System.out.println("Please enter y: ");
      y = sc.nextInt();
      if (x == 0.0)
         result = 0.0;
      else if (y < 0)
         while (y != 0) {
            result *= 1/x;  y++;  // two statements here
         }
      else
         while (y != 0) {
            result *= x;    y--;
         }
      System.out.println("Result is " + result);
} }
```

Program Execution

Inputs/initialization:

(i) **x**= 2, **y** =-3, **result** = 1.0 **(ii)** **x**= 2, **y** =3, **result** = 1.0

y	y != 0	result (*= 1/x)
-3	true	1*(½)=½
-2	true	½*(½) = ¼
-1	true	¼*(½) = 1/8
0	false	

y	y != 0	result (*= x)
3	true	1*(2)=2
2	true	2*(2) = 4
1	true	4*(2) = 8
0	false	

Outputs:

Result = 1/8 Result = 8

Program Input and Output

```
Please enter x:
2
Please enter y :
3
Result is 8.0
```

```
Please enter x:
2
Please enter y :
0
Result is 1.0
```

```
Please enter x:
2
Please enter y :
-3
Result is 0.125
```

Review Questions

What output is produced by the following code fragment? How many times is the loop body repeated?

```
int num=11, max=10;
while (num < max)
{
  if (num%2 == 0)
    System.out.println(num);
  num++;
}
```

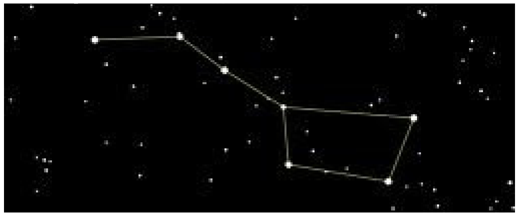
```
Suppose num = 11?
Suppose num = 1?
```

Note:

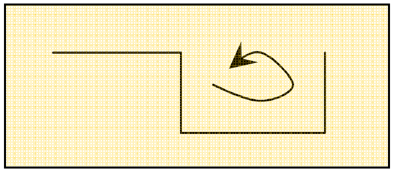
- Trace all examples in this chapters (and more if help) to understand the logic!!!
- A usual mistake is that after you write a loop, your code results in one additional OR one less iteration than it should be

Looping

- Why Loops?
- The while Statement
- **The for Statement**
- The do while Statement
- The break and continue Statement
- Nested Loops
- Case Study



The for Loop

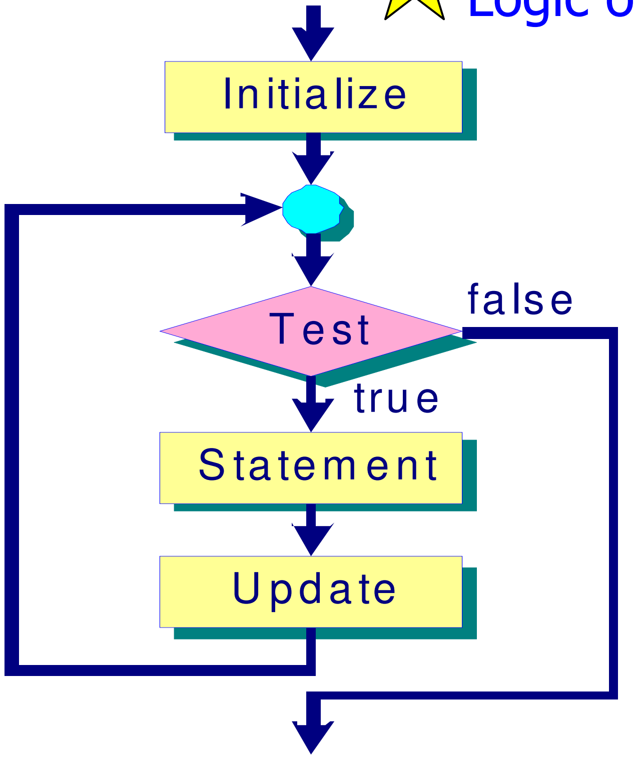


All repetition logic can be written using while loops.
The for statement is another way of writing repetition logic.
The for statement aims to handle **counter-controlled loops**.

for (Initialize; Test; Update)
Statement;

Statement can be a simple statement terminated by a semicolon or a compound statement enclosed by { }

Logic of for loop



- Normally, *Test* is a relational expression to control iterations

- *Update* is frequently used to update some loop control variables before repeating the loop

- Any or all of the 3 expressions may be omitted. In case test is missing, it becomes an infinite loop, i.e. all statements inside the loop will be executed again and again. For example,

```
for (;;) {     /* an infinite loop */
        statement1;
        ...
}
```

- Example: Counter-Controlled Loop

Compare

```
int counter=0;
double total=0.0;
while (counter < 50)
{
    System.out.print(“Enter mark: “);
    double mark = sc.nextDouble();
    total += mark;
    counter++;
}
```

```
int counter;
double total=0.0;
for (counter=0; counter < 50; counter++)
{
  System.out.print(“Enter mark: “);
  double mark = sc.nextDouble();
  total += mark;
}
```

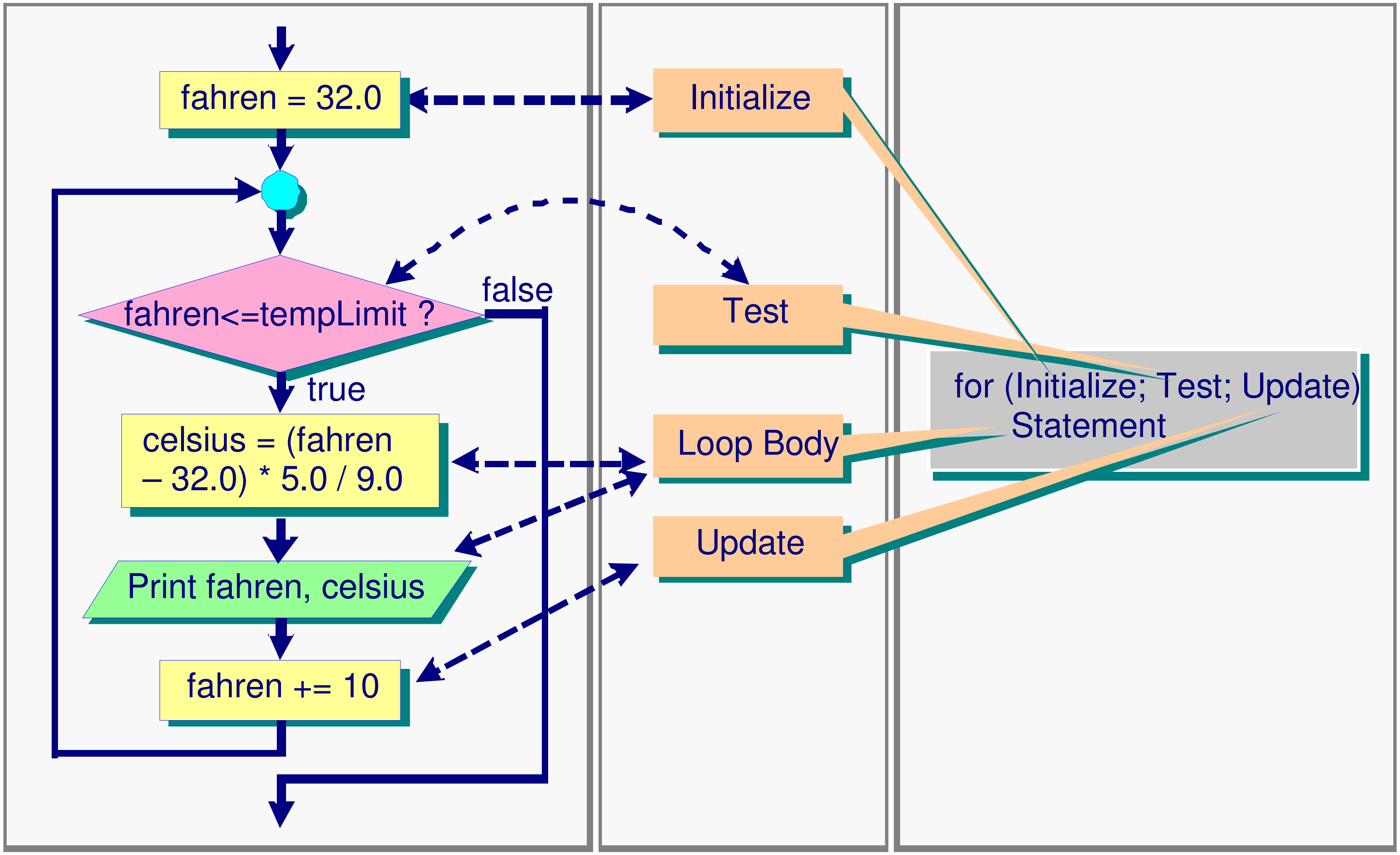
more organized
and readable

Example: Computing Temperature

```
import java.util.Scanner;
public class ConvertTemp2 {
   public static void main(String[] args) {
      double fahren, celsius;
      double tempLimit;
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter conversion limit (F): ");
      tempLimit = sc.nextDouble();
      System.out.println("\tFahrenheit\tCelsius");
      System.out.println("\t----------\t-------");

      for (fahren=32.0; fahren<=tempLimit; fahren+=10)
      {
         celsius = (fahren - 32.0) * 5.0/9.0;
         System.out.println("\t " + fahren + "\t\t\t"
                   + celsius);
      }
   }
}
```


fahren = 32.0
Initialize
fahren<=tempLimit ?
false
Test
true
celsius = (fahren – 32.0) * 5.0 / 9.0
Loop Body
for (Initialize; Test; Update)
Statement
Print fahren, celsius
Update
fahren += 10

Example: Summing a Series of Data using for Loop

$$1-\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-\cdots\cdots+\frac{x^{10}}{10!}$$

for loop to visit each term

(variables)

term -> x^n
denom -> n!
sign -> +/-

Add to temp

```
import java.util.Scanner;
public class SumSeriesData {
   public static void main(String[] args) {
      double x, temp = 1.0, term = 1.0;
      int n, sign = 1, denom = 1;
      Scanner sc = new Scanner(System.in);
      System.out.println("Please enter the value of x:");
      x = sc.nextDouble();
      for (n = 1; n <= 10; n++) {
            denom *=  n     ;
            sign   = -sign ;
            term  *=  x     ;
            temp  += sign * term / denom ;
      }
      System.out.println( "The result is " + temp );
   }
}
```

Accumulate!!!

Program Execution

Inputs/initialization:

x = 0.9, **temp** = 1.0, **term** = 1.0

n	**n <= 10**	**denom** **(*= n)**	**sign**	**term** **(*=x)**	**temp** **(+=** **sign*term/denom)**
1	true	1	-1	0.9	0.0999
2	true	2	+1	0.81	0.505
…	…	…	…		
10	true	3628800		0.3486	0.406569
11	**false**				

Outputs:

Result = 0.406569

Looping

- Why Loops?
- The while Statement
- The for Statement
- **The do while Statement**
- The break and continue Statement
- Nested Loops
- Case Study

The do-while Loop

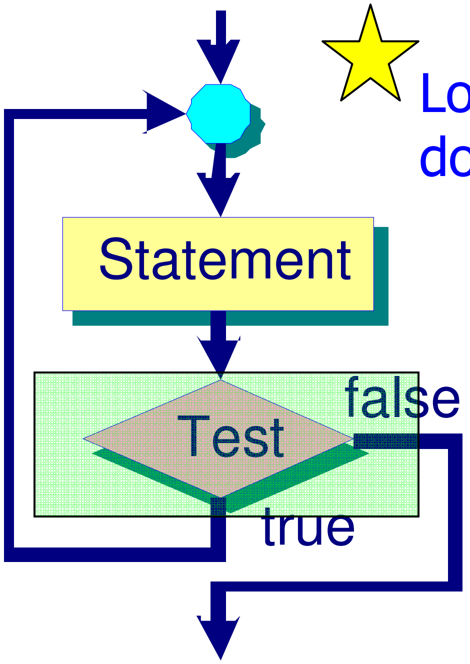
For **while** and **for** loops

1. **test for conditions** and then
2. when the condition is true, then execute the statements in the loop.

How about **do-while** loop? It is similar to the **while** statement.

```
do
    Statement;
while (Test);
```

Statement can be a simple statement terminated by a semicolon or a compound statement enclosed by { }



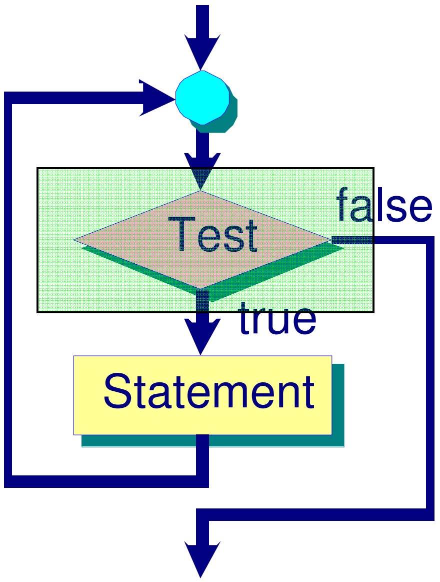
Logic of do-while loop

- It differs from the ***for*** and ***while*** statements in that the condition test, i.e. *Test*, is performed **after** executing the statement every time.

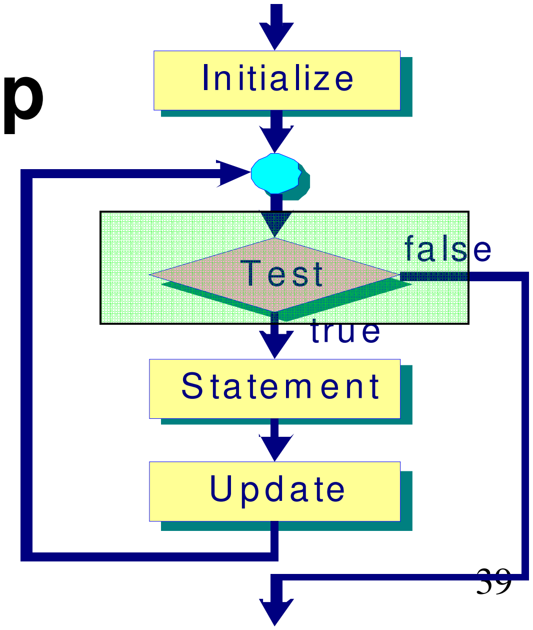
=> This means the loop will be executed **at least once**!!!!!

=> On the other hand, the *while* or *for* loop might **not** be executed even once.

while loop



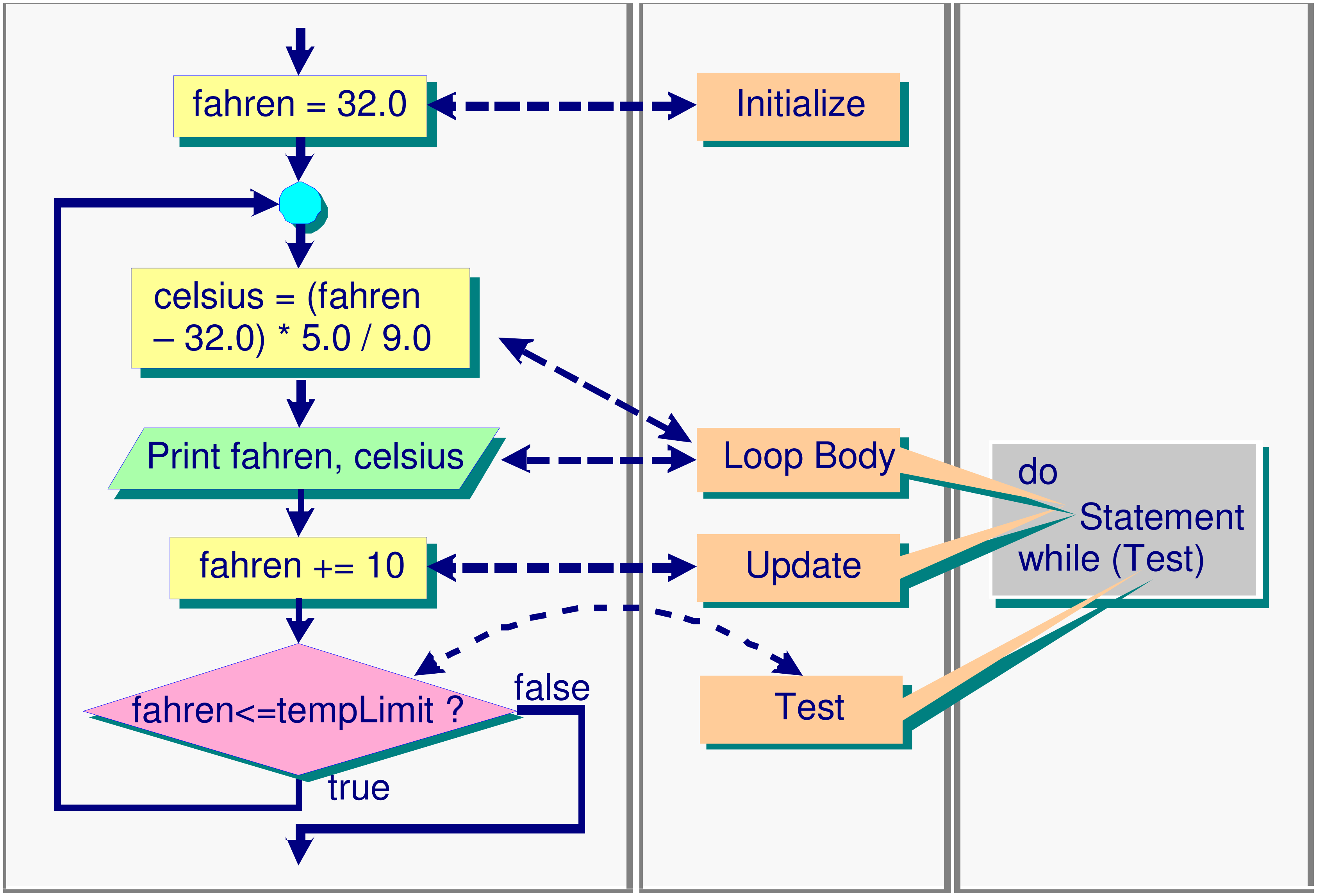
for loop



Example: Computing Temperature

```
import java.util.Scanner;
public class ConvertTemp3 {
   public static void main(String[] args) {
      double fahren, celsius;
      double tempLimit;
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter conversion limit (F): ");
      tempLimit = sc.nextDouble();
      System.out.println("\tFahrenheit\tCelsius");
      System.out.println("\t----------\t-------");
  1   fahren = 32.0;
      if (fahren <= tempLimit)
        do {
   3        celsius = (fahren - 32.0) * 5.0/9.0;
            System.out.println("\t " + fahren + "\t\t\t"
  4               + celsius);
  2         fahren += 10;
        } while (fahren <= tempLimit);
   }
}
```


fahren = 32.0
celsius = (fahren – 32.0) * 5.0 / 9.0
Print fahren, celsius
fahren += 10
fahren<=tempLimit ?
false
true
Initialize
Loop Body
Update
Test
do
Statement
while (Test)

Example: Using the do-while Loop

```
import java.util.Scannerl
public class UsingDoWhile {
   public static void main(String[] args) {
      int input;        /* User input number. */
      Scanner sc = new Scanner(System.in);
      do {
         System.out.println("Input a number(1 to 5): ");
         input = sc.nextInt();
         if (input > 5 || input < 1) {
            System.out.print(input+ "is out of range! ");
            System.out.println("Try again.");
         }
      } while (input > 5 || input < 1);
      System.out.println("Input = " + input);
   }
}
```

Program Input and Output

```
Input a number (1 to 5): 0
0 is out of range! Try again.
Input a number (1 to 5): 6
6 is out of range! Try again.
Input a number (1 to 5): 5
Input = 5
```

Which loop do we use?

- **for loop** – most appropriate for a fixed number of repetition (i.e. **counter-**controlled loops).

- **while loop** – most appropriate for **sentinel**-controlled loops.

- **do-while loop** – most appropriate for applications that loop at least once. For example, **menu display and selection**.

Review Questions

What is the difference between while and do while loop?

What output is produced by the following code fragment?

```
for (int num = 10; num >= 0; --num);
  if (num%4 == 0)
     System.out.println(num);
```

Convert the following for loop into while loop and do while loop??

```
int sum=0;
for (int i=0; i<=10; i++)
  sum = sum + 1;
```

Looping

- Types of Loops
- The while Statement
- The for Statement
- The do while Statement
- **The break and continue Statement**
- Nested Loops
- Case Study

The break Statement

- The ***break*** statement can alter the control flow inside the *switch* statement and loops, i.e. *while*, *for*, ..., etc.

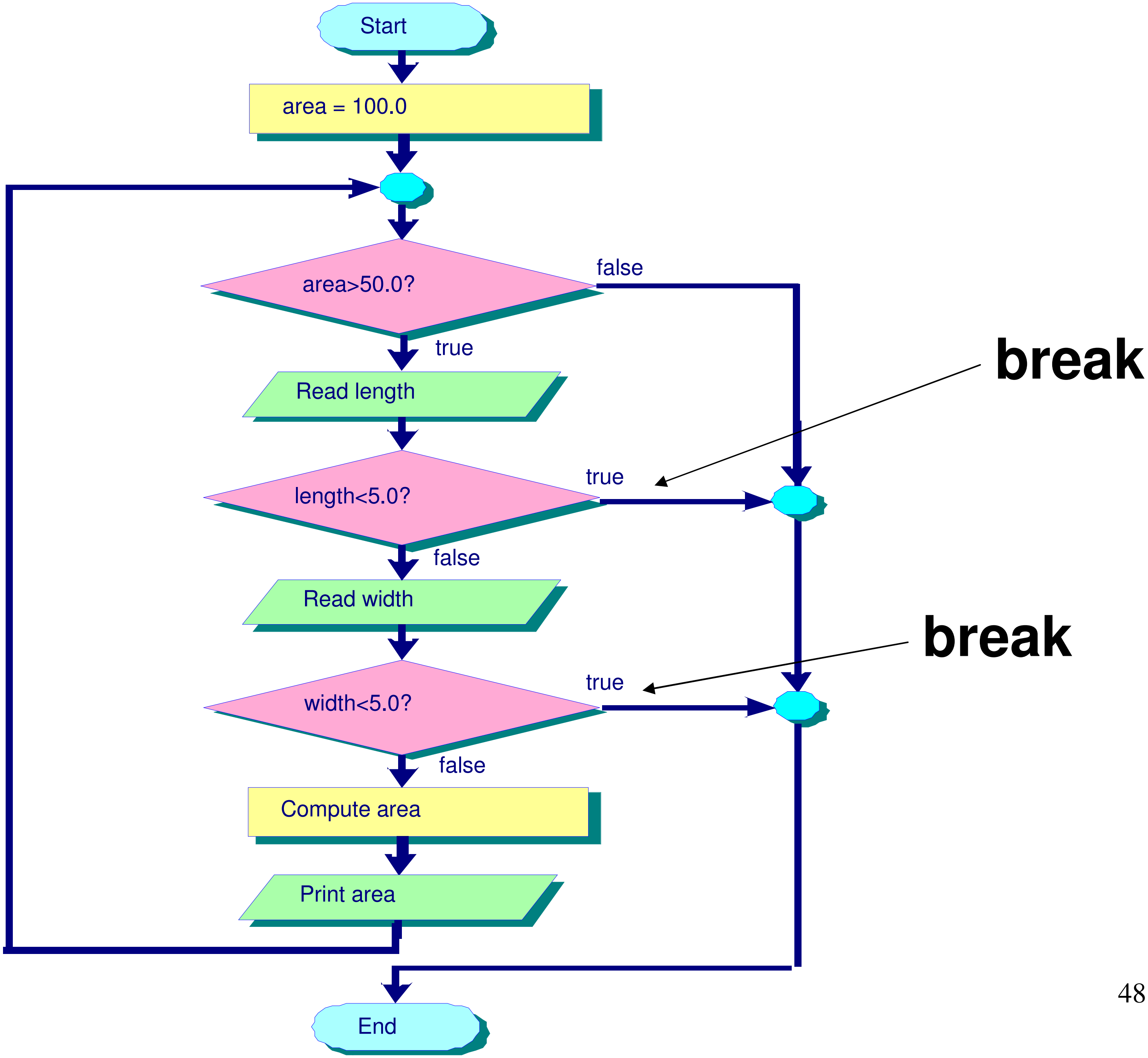
- Execution of ***break*** causes **immediate termination** of the **innermost** enclosing loop or the switch statement.

Example: Using the break Statement

```
import java.util.Scanner;
public class UsingBreak {
   public static void main(String[] args) {
      double length, width;
      double area=100.0;
      Scanner sc = new Scanner(System.in);
      while (area > 50.0) {
        System.out.println("Enter length of rect: ");
        length = sc.nextDouble();
        if (length < 5.0)
            break;
        System.out.println("Enter width of rect: ");
        width = sc.nextDouble();
        if (width < 5.0)
            break;
        area = length*width;
        System.out.println("The area = " + area);
      }
   }
}
```

Program Input and Output

```
Enter length of rect: 10.0
Enter width of rect: 20.0
The area = 200.0
Enter length of rect: 50.0
Enter width of rect: 4.0
```


Start
area = 100.0
area>50.0?
false
true
Read length
length<5.0?
true
false
Read width
width<5.0?
true
false
Compute area
Print area
End
break
break

The Continue Statement

- The ***continue*** statement can only alter the flow **inside** a loop.

- Complements the **break** statement.

- Its use is restricted to *while*, *for*, and *do while* loop.

- Control immediately **passed to**
  1) test condition of the **nearest** enclosing while/do-while
  2) update of the for loop

  i.e., all subsequent statements after the continue statement are **not** executed for **this** particular iteration.

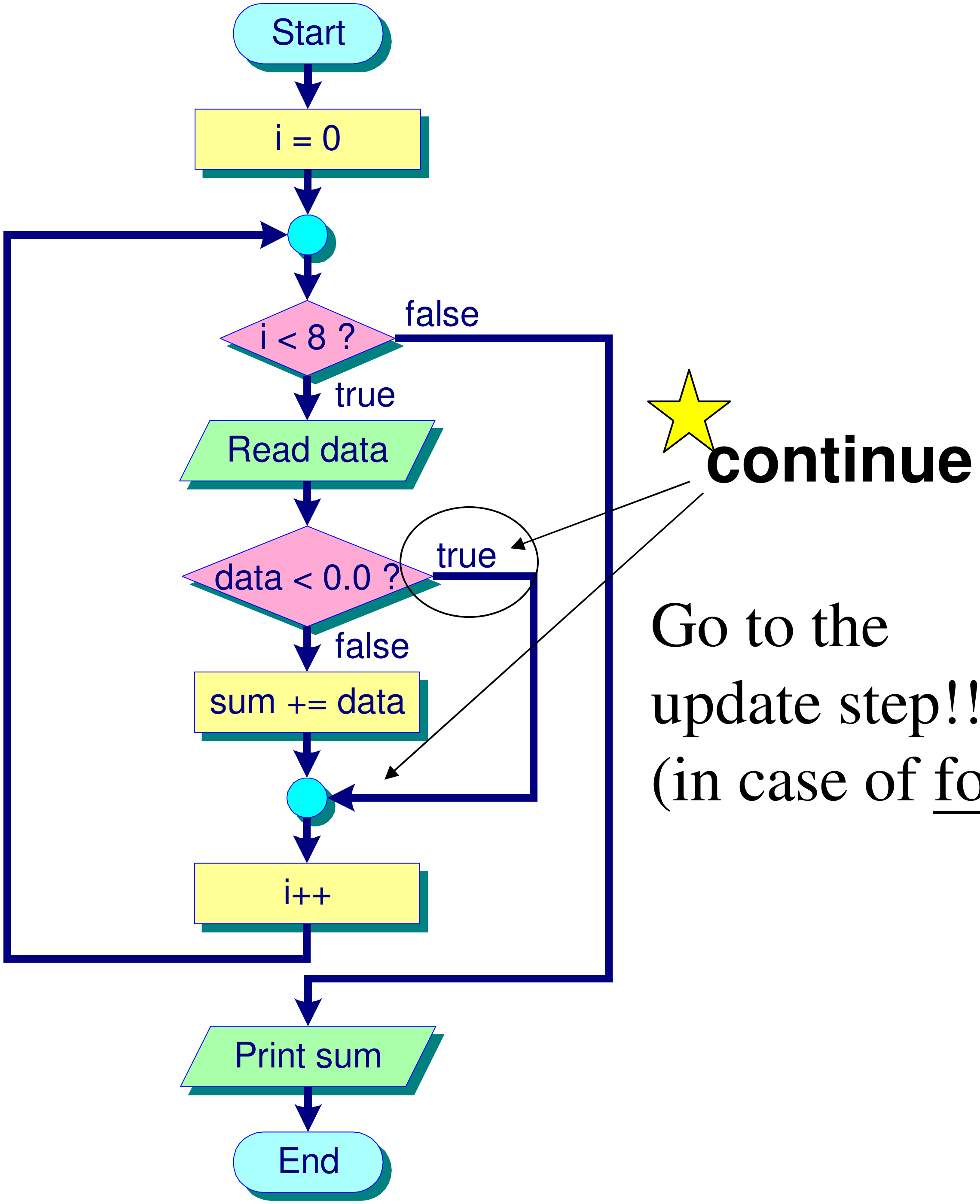
Example: Using the Continue Statement

```
import java.util.Scanner;
public class SumPosNumbers {
   public static void main(String[] args) {
      int i;
      double data, sum = 0;
      Scanner sc = new Scanner(System.in);
      // Read 8 numbers
      System.out.println("Enter 8 numbers: ");
      for (i = 0; i < 8; i++) {
            data = sc.nextDouble();
            if (data < 0.0)
               continue;        // go to update: i++
            sum += data;
      }
      System.out.println("The sum is " + sum);
   }
}
```

Program Input and Output

Enter 8 numbers: *1* *2* *3* *4* ~~*-5*~~ *6* ~~*-7*~~ *8*

The sum is 24.0



Start

i = 0

i < 8 ?

false

true

Read data

data < 0.0 ?

true

false

sum += data

i++

Print sum

End

continue

Go to the
update step!!!!!!!
(in case of <u>for loop</u>)

Program Execution

Inputs/initialization:

data = 1, 2, 3, 4, -5, 6, -7, 8

i	i < 8	data	data < 0.0	sum
0	true	1	false	1
1	true	2	false	3
2	true	3	false	6
3	true	4	false	10
4	true	-5	**true**	
5	true	6	false	16
6	true	-7	**true**	
7	true	8	false	24
8	**false**			

Outputs:

Average = 24.0

Review Questions

What output is produced by the following code fragment?

(A)

```
int x = 10;
while (true) {
  if (x < 5)
     break;
  x = x - 2;
}
System.out.println(“x is ” + x);
```

OK??

(B)

```
int x = 10;
while (true) {
  if (x < 5)
     continue;
  x = x - 2;
}
System.out.println(“x is ” + x);
```

OK??

Chapter 7
Looping

- Types of Loops
- The while Statement
- The for Statement
- The do while Statement
- The break and continue Statement
- **Nested Loops**
- Case Study

Nested Loops

- A loop may appear inside another loop. This is called a **nested loop**.

- We can nest as **many levels** of loops as the hardware allows.

- And we can nest **different types** of loops.

- Two useful applications:
  1. Printing 2-D tables
  2. Printing patterns (also 2-D)

Example: Printing Multiplication Table

Column j

Row i

	1	2	3	…	9
1	1	2	3		9
2	2	4	6		18
3	3	6	9		27
4	4	8	12		36
…					…
9	9	18	27		81

Example: Printing Multiplication Table

Nested Loop using while

Trace it!!!!!

```
int i=1;
while (i <= 9) {
    int j=1;
    System.out.println(i + “Multiplication Table”);
    while (j <= 9) {
        System.out.println(i + “ x ” +j+ “ = “ + (i*j));
        j++;
    }
    i++;
}
```

i=1	j=1	1x1 = 1
	j=2	1x2 = 2
	…	…
i=2	j=1	2x1 = 2
	j=2	2x2 = 4
	…	…
i=9	i=1	9x1 = 9
	i=2	9x2 = 18
	…	…

Example: Printing Multiplication Table

Nested Loop using for

Trace it!!!!!

```
for (int i = 1; i <= 9; i++) {
    System.out.println(i + “Multiplication Table”);
    for (int j=1; j<=9; j++)
        System.out.println(i+ “ x ” +j+ “ = “+ (i*j));
}
```

Nested Loop using do-while

Trace it!!!!!

```
int i=1;
do {
   int j=1;
   System.out.println(i + “Multiplication Table”);
   do {
      System.out.println(i + “ x ” +j+ “ = “ + (i*j));
      j++;
   } while (j <= 9);
   i++;
} while (i <= 9);
```

Example: Printing Triangular Shape (2D pattern)

Trace it!!!!!

```
import java.util.Scanner;
public class PatternLoopApp {
   public static void main(String[] args) {
      int a, b, height, lines;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter height of a pattern: ");
      height = sc.nextInt();
      for (lines = 1; lines <= height; lines++) {
         for (a = 1; a <= (height-lines); a++)
            System.out.print(' ');
         for (b = 1; b <= (2*lines - 1); b++)
            System.out.print('*');
         System.out.println();
      }
   }
}
```

Program Input and Output

```
Enter the height of a pattern: 5
    *
   ***
  *****
 *******
*********
```

a,b

lines

Program Execution

Trace it!!!!!

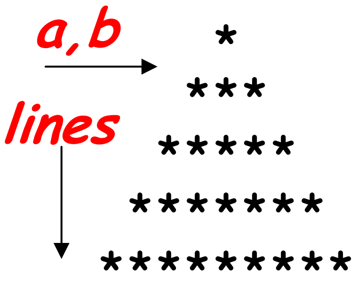
lines=1
println
	a=1
	a=2
	a=3
	a=4
	b=1
		_
		_ _
		_ _ _
		_ _ _ _
		_ _ _ _ *

lines=2
println
	a=1,..,3
	b = 1,..,3
		_ _ _
		_ _ _ ***

lines=3
lines=4
lines=5
	….	_ _ *****
		_ *******
		*********

```
for (lines = 1; lines <= height; lines++) {
   for (a = 1; a <= (height-lines); a++)
      System.out.print(' ');
   for (b = 1; b <= (2*lines - 1); b++)
      System.out.print('*');
   System.out.println();
}
```

Program Input and Output



```
Height: 5
    *
   ***
  *****
 *******
*********
```

Review Exercise: Printing patterns

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

```
    1
   131
  13531
 1357531
135797531
 1357531
  13531
   131
    1
```

How about these patterns?

Review Questions

What output is produced by the following code fragment?

(A)

```
for (int i=1; i<5; i++){
  System.out.println(“i = “ + i);
  for (int j=1; j<5; j++){
     if (i*j < 5)
        break;
     System.out.println(i*j);
  }
}
```

Trace Carefully!!!!

(B)

```
for (int i=1; i<5; i++){
  System.out.println(“i = “ + i);
  for (int j=1; j<5; j++){
     if (i*j < 5)
        continue;
     System.out.println(i*j);
  }
}
```

Looping

- Types of Loops
- The while Statement
- The for Statement
- The do while Statement
- The break and continue Statement
- Nested Loops
- **Case Study**

Case Study: Generating Statistics of Student Marks

Problem Specification

Computer engineering students are required to take the course on Introduction to Java Programming in their first year of study. The students are required to take an examination at the end of the course. In order to know how the students perform during the examination, you are asked to write a program to generate the statistics of the examination results.

→ **purpose**

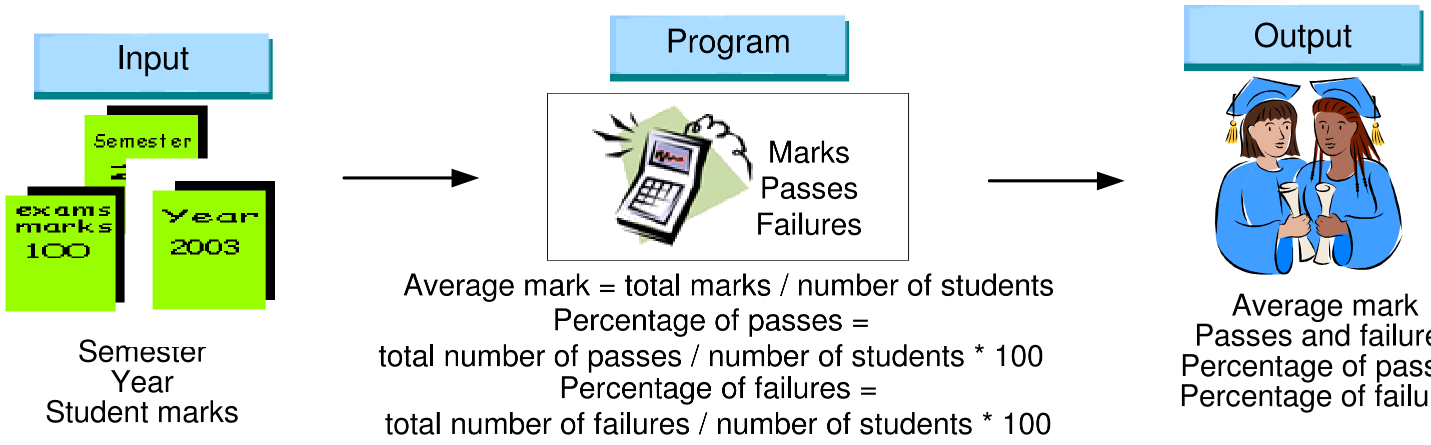
You will be given a list of marks for all the students who have taken the course in a specified semester and year.

→ **input**

The program should find the average mark, the number of passes and failures, and the percentage of passes and failures of the course. The passing mark is 50.

→ **output**

Problem Analysis



Input

Semester
Year
Student marks

Program

Marks
Passes
Failures

Average mark = total marks / number of students
Percentage of passes =
total number of passes / number of students * 100
Percentage of failures =
total number of failures / number of students * 100

Output

Average mark
Passes and failures
Percentage of passes
Percentage of failures

Required inputs:

- the semester and year
- a list of marks

- DO NOT KNOW THE NUMBER OF STUDENTS

Required output:

- the average mark
- the number of passes and failures
- the percentage of passes and failures

Formulas:

- average mark = total marks/number of students
- percentage of passes = (total num of passes/num of students) * 100
- percentage of failures = (total num of failures/num of students) * 100

Program Design

Initial Algorithm

1. Read semester, year and initialize variables.
2. For each student, input the mark, process the mark to update total marks, the number of passes and failures, and number of students. This process will stop when the input ends.
3. Compute the statistics on average, percentage of passes and failures.
4. Print the statistics for semester, year, average, passes, failures, percentage of passes, percentage of failures.

Program Design

Algorithm in Pseudocode

```
main:
  READ semester, year
  SET totalMarks, passes, failures TO 0
  READ mark
  WHILE mark is not equal to -1
    ADD mark TO totalMarks
    ADD 1 TO numOfStudent
    IF mark < 50
      ADD 1 TO failures
    ELSE
      ADD 1 TO passes
    ENDIF
    READ mark
  ENDWHILE
  COMPUTE average = totalMarks/numOfStudent
  COMPUTE percentPass = (passes/numOfStudent) * 100
  COMPUTE percentFail = (failures/numOfStudent) * 100
  PRINT semester, year, average, passes, failures,
   percentPass, percentFail
```

- USE SENTINEL CONTROLLED LOOP

Program Design

Program Dry-run

Inputs:

semester = 2, **year** = 2005, **mark** = 80, 75, 50, 20, -1

mark	mark not equal to -1	numOf Student	total Marks	failures	passes
80	true	1	80	0	1
75	true	2	155	0	2
50	true	3	205	0	3
20	true	4	225	1	3
-1	**false**				

Outputs:

The average mark = 56.25

The number of passes = 3

The number of failures = 1

The passing rate = 75.0%

The failure rate = 25.0%

Implementation

```
import java.text.*; import java.util.Scanner;
public class StudentStatApp {
   public static void main(String[] args) {
      double average, percentPass, percentFail, mark,
         totalMarks;
      int semester, year;
      int numOfStudent = 0;
      int passes,failures; Scanner sc= new Scanner(System.in);
      DecimalFormat numForm= new DecimalFormat("###.00");
      // Read semester and year and initialize variables
      System.out.println("Enter the Semester : ");
      semester = sc.nextInt();
      System.out.println("Enter the Year: ");
      year = sc.nextInt();
      totalMarks = passes = failures = 0;
      // Input and process the marks
      mark = 0.0;
      System.out.println("Enter the mark(-1 to end): ");
      mark = sc.nextInt();
      while (mark != -1)
      {
         totalMarks += mark;
         numOfStudent++;
```

```
     if  (mark < 50)
        failures++;
      else
        passes++;
       System.out.println("Enter mark (-1 to end): ");
      mark = ConsoleIn.readlnInt();
   }
   // Compute the statistics
  average =(double)totalMarks/(double)numOfStudent;
  percentPass =(double)passes/(double)numOfStudent*100;
  percentFail=(double)failures/(double)numOfStudent*100;
  // Print the statistics
  System.out.println("For semester " + semester +
      " and year " + year);
  System.out.println("The average mark = " +
      numForm.format(average));
  System.out.println("The number of passes = " + passes);
  System.out.println("The number of failures = " + failures);
  System.out.println("The passing rate  = " +
      numForm.format(percentPass) + "%");
  System.out.println("The failure rate  = " +
      numForm.format(percentFail) + "%");
  }
}
```

Testing

Program input and output

```
Enter the Semester:
2
Enter the Year:
2004
Enter mark (-1 to end):
80
Enter mark (-1 to end):
75
Enter mark (-1 to end):
50
Enter mark (-1 to end):
20
Enter mark (-1 to end):
-1
```

- USING SENTINEL VALUE OF -1

```
For semester 2 and year 2004
The average mark = 56.25
The number of passes = 3
The number of failures = 1
The passing rate = 75.00%
The failure rate = 25.00%
```

Key Terms

- loop
- loop body
- counter-controlled loop
- sentinel-controlled loop
- sentinel value
- infinite loop
- break statement
- continue statement
- nested loop

Further Reading

- Read Chapter 7 on “Looping” of the textbook.
- Read other case studies from the chapter.