APROG – Algoritmia e Programação

Emanuel Cunha Silva
ecs@isep.ipp.pt

Bibliography:
Big Java Late Objects
Book by Cay S. Horstmann

Chapter Goals

- To implement `while`, `for`, and `do` loops
- To hand-trace the execution of a program
- To become familiar with common loop algorithms
- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

The `while` Loop

- Examples of loop applications
  - Calculating compound interest
  - Simulations, event driven programs…
- Compound interest algorithm (Chapter 1)

Start with a year value of 0, a column for the interest, and a balance of $10,000.

year	interest	balance
0		$10,000

Steps

Repeat the following steps while the balance is less than $20,000.

Add 1 to the year value.

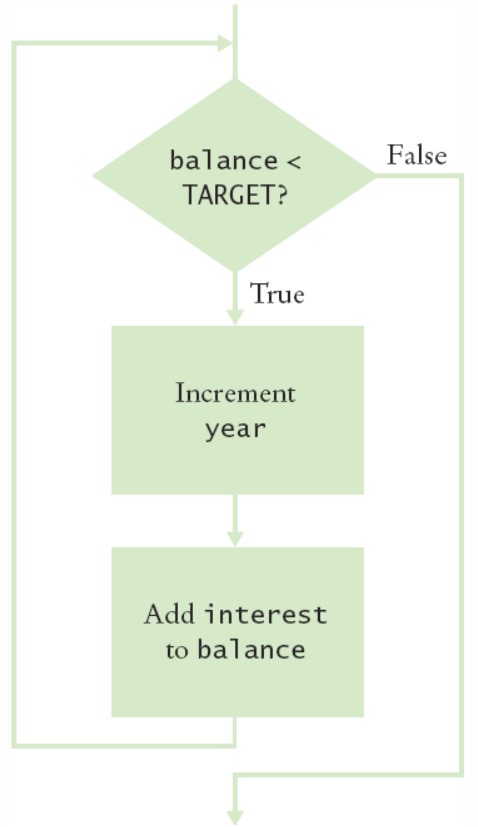
Compute the interest as balance x 0.05 (i.e, 5 percent interest).

Add the interest to the balance.

Report the final year value as the answer.

Planning the `while` Loop

- A loop executes instructions repeatedly while a condition is true




```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE/100;
   balance = balance + interest;
}
```

while Statement

Syntax

```
while (condition)
{
   statements
}
```

This variable is declared outside the loop and updated in the loop.

Beware of "off-by-one" errors in the loop condition.

If the condition never becomes false, an infinite loop occurs.

Don't put a semicolon here!

```
double balance = 0;
.
.
.
while (balance < TARGET)
{
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

These statements are executed while the condition is true.

This variable is created in each loop iteration.

Lining up braces is a good idea.

Braces are not required if the body contains a single statement, but it's good to always use them.

Execution of the Loop

1 Check the loop condition

balance =	10000
year =	0

The condition is true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

2 Execute the statements in the loop

balance =	10500
year =	1
interest =	500

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

3 Check the loop condition again

balance =	10500
year =	1

The condition is still true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

4 After 15 iterations

balance =	20789.28
year =	15

The condition is no longer true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

5 Execute the statement following the loop

balance =	20789.28
year =	15

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
System.out.println(year);
```

DoubleInvestment.java

```java
/**
   This program computes the time required to double an investment.
*/
public class DoubleInvestment
{
   public static void main(String[] args)
   {
      final double RATE = 5;
      final double INITIAL_BALANCE = 10000;
      final double TARGET = 2 * INITIAL_BALANCE;

      double balance = INITIAL_BALANCE;
      int year = 0;

      // Count the years required for the investment to double

      while (balance < TARGET)
      {
         year++;
         double interest = balance * RATE / 100;
         balance = balance + interest;
      }

      System.out.println("The investment doubled after "
         + year + " years.");
   }
}
```

Declare and initialize a variable outside of the loop to count `years`

Increment the `years` variable each time through

Program Run

```
The investment doubled after 15 years.
```

while Loop Examples (1)

Loop	Output	Explanation

`i = 0; sum = 0;`
`while (sum < 10)`
`{`
`  i++; sum = sum + i;`
`  Print i and sum;`
`}`
	1 1
	2 3
	3 6
	4 10
		When sum is 10, the loop condition is false, and the loop ends.

`i = 0; sum = 0;`
`while (sum < 10)`
`{`
`  i++; sum = sum - i;`
`  Print i and sum;`
`}`
	1 -1
	2 -3
	3 -6
	4 -10
	. . .
		Because sum never reaches 10, this is an "infinite loop"

`i = 0; sum = 0;`
`while (sum < 0)`
`{`
`  i++; sum = sum - i;`
`  Print i and sum;`
`}`
	(No output)	The statement sum < 0 is false when the condition is first checked, and the loop is never executed.

while Loop Examples (2)

Loop	Output	Explanation

`i = 0; sum = 0;`
`while (sum >= 10)`
`{`
`  i++; sum = sum + i;`
`  Print i and sum;`
`}`
	(No output)	The programmer probably thought, "Stop when the sum is at least 10." However, the loop condition controls when the loop is executed, not when it ends

`i = 0; sum = 0;`
`while (sum < 10) ;`
`{`
`  i++; sum = sum + i;`
`  Print i and sum;`
`}`
	(No output, program does not terminate)	Note the semicolon before the `{`. This loop has an empty body. It runs forever, checking whether `sum < 0` and doing nothing in the body.

Self Check

What does the following loop print?

```
int n = 1;
while (n < 100)
{
   n = 2 * n;
   System.out.print(n + " ");
}
```

Answer: `2 4 8 16 32 64 128`

Note that the value 128 is printed even though it is larger than 100.

Common Error

- Don't think "Are we there yet?"
  - The loop body will only execute if the test condition is **True**
  - "Are we there yet?" should continue if **False**
  - If `bal` should grow until it reaches `TARGET`
    - Which version will execute the loop body?

```
while (bal < TARGET)
{
   year++;
   interest = bal * RATE;
   bal = bal + interest;
}
```

```
while (bal >= TARGET)
{
   year++;
   interest = bal * RATE;
   bal = bal + interest;
}
```

Common Error

- Infinite Loops
  - The loop body will execute until the test condition becomes **False**
  - What if you forget to update the test variable?
    - `bal` is the test variable (`TARGET` doesn't change)
    - You will loop forever! (or until you stop the program)

```
while (bal < TARGET)
{
   year++;
   interest = bal * RATE;
}
```

Common Error

- Off-by-One Errors
  - A ‘counter’ variable is often used in the test condition
  - Your counter can start at 0 or 1, but programmers often start a counter at 0
  - If I want to paint all 5 fingers, when I am done?
    - **Start at 0, use <**

```
int finger = 0;
final int FINGERS = 5;
while (finger < FINGERS)
{
   // paint finger
   finger++;
}
```

Start at 1, use <=

```
int finger = 1;
final int FINGERS = 5;
while (finger <=
    FINGERS)
{
   // paint finger
   finger++;
}
```

Problem Solving: Hand-Tracing

- Example: Calculate the sum of digits (1+7+2+9)
  - Make columns for key variables (`n`, `sum`, `digit`)
  - Examine the code and number the steps
  - Set variables to state before loop begins

n	sum	digit
1729	0	

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Start executing loop body statements changing variable values on a new line

- Cross out *values* in previous line

n	sum	digit
1729	~~0~~	
	9	9

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Continue executing loop statements changing variables

- 1729 / 10 leaves 172 (no remainder)

n	sum	digit
~~1729~~	~~0~~	
172	9	9

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Test condition. If true, execute loop again
  - Variable `n` is 172, Is 172 > 0?, True!

- Make a new line for the second time through and update variables

n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
17	11	2

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Third time through
  - Variable `n` is 17 which is still greater than 0

- Execute loop statements and update variables

n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
1	18	7

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Fourth loop iteration:
  - Variable `n` is 1 at start of loop. 1 > 0? True
  - Executes loop and changes variable n to 0 (1/10 = 0)

n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
~~1~~	~~18~~	~~7~~
0	19	1

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Because `n` is 0, the expression (`n` > 0) is False
- Loop body is not executed
  - Jumps to next statement after the loop body
- Finally prints the sum!

n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
~~1~~	~~18~~	~~7~~
0	19	1

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

Self Check

Hand-trace the following code, showing the value of `n` and the output.

```
int n = 5;
while (n >= 0)
{
   n--;
   System.out.print(n);
}
```

Answer:

n	output
~~5~~	
~~4~~	4
~~3~~	3
~~2~~	2
~~1~~	1
~~0~~	0
-1	-1

Self Check

Hand-trace the following code, showing the value of `n` and the output. What potential error do you notice?

```
int n = 1;
while (n <= 3)
{
   System.out.print(n + ", ");
   n++;
}
```

Answer:

n	output
~~1~~	1,
~~2~~	1, 2,
~~3~~	1, 2, 3,
4	

There is a comma after the last value. Usually, commas are between values only.

Self Check

Hand-trace the following code, assuming that `a` is 2 and n is 4. Then explain what the code does for arbitrary values of `a` and `n`.

```
int r = 1;
int i = 1;
while (i <= n)
{
   r = r * a;
   i++;
}
```

Answer:

a	n	r	i
2	4	~~1~~	~~1~~
		~~2~~	~~2~~
		~~4~~	~~3~~
		~~8~~	~~4~~
		16	5

The code computes a^n.

Self Check

Trace the following code. What error do you observe?

```
int n = 1;
while (n != 50)
{
   System.out.println(n);
   n = n + 10;
}
```

Answer:

n	output
~~1~~	1
~~11~~	11
~~21~~	21
~~31~~	31
~~41~~	41
~~51~~	51
~~61~~	61
...	

This is an infinite loop. `n` is never equal to 50.

Self Check

The following pseudocode is intended to count the number of digits in the number n:

```
count = 1
temp = n
while (temp > 10)
  Increment count.
  Divide temp by 10.0.
```

Trace the pseudocode for n = 123 and n = 100. What error do you find?

Answer:

count	temp
1	123
2	12.3
3	1.23

This yields the correct answer. The number 123 has 3 digits.

count	temp
1	100
2	10.0

This yields the wrong answer. The number 100 also has 3 digits. The loop condition should have been

```
while (temp >= 10)
```

The for Loop

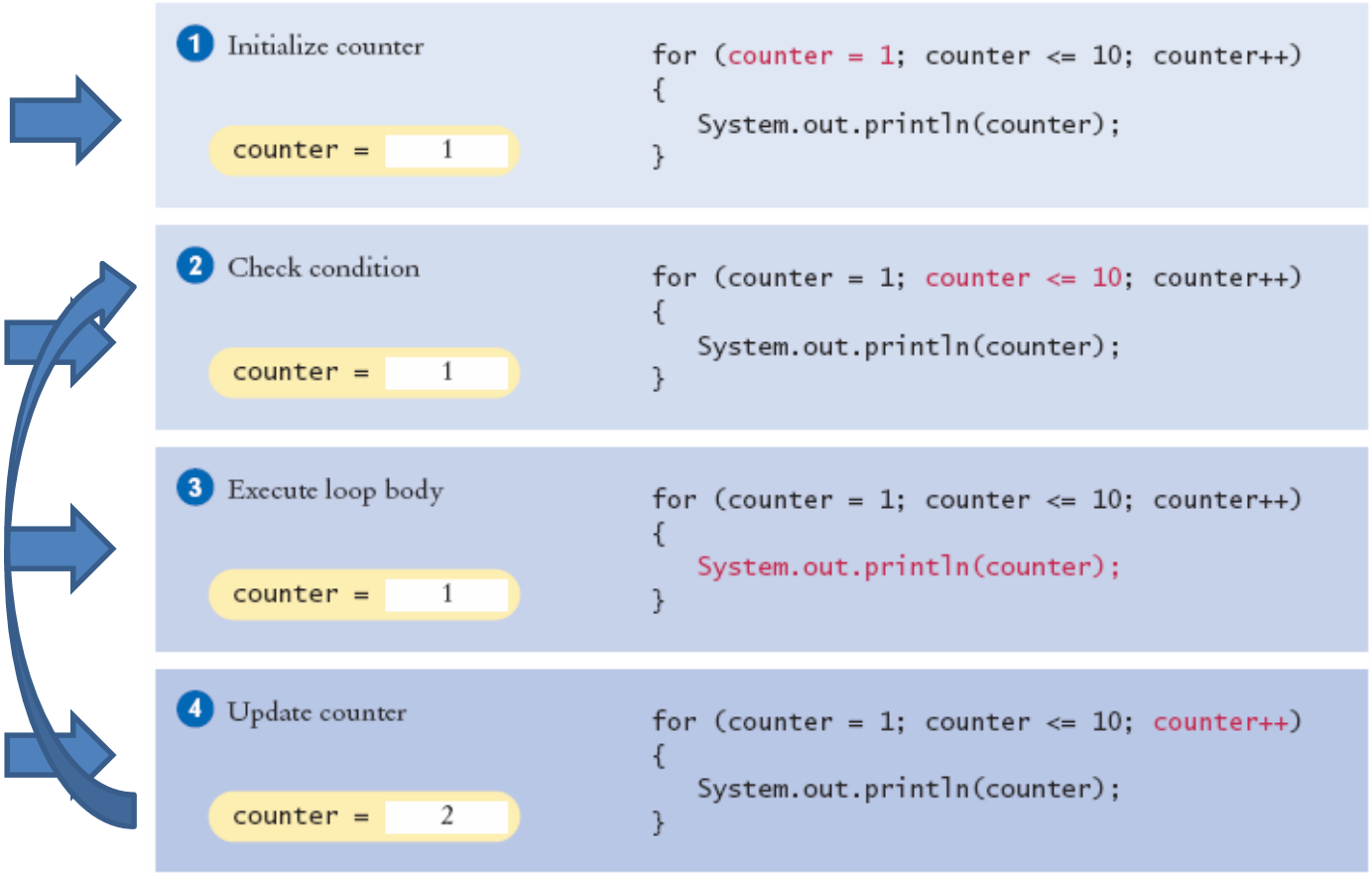
- Use a `for` loop when you:
  - Can use an integer counter variable
  - Have a constant increment (or decrement)
  - Have a fixed starting and ending value for the counter

- Use a `for` loop when a value runs from a starting point to an ending point with a constant increment or decrement

```
int i = 5;  // initialize
while (i <= 10)  // test
{
    sum = sum + 1;
    i++; // update
}
```

```
for (int i = 5; i <= 10; i++)
{
    sum = sum + i;
}
```

Execution of a for Loop



for Statement

Syntax

```
for (initialization; condition; update)
{
   statements
}
```

These three expressions should be related.

This initialization happens once before the loop starts.

The condition is checked before each iteration.

This update is executed after each iteration.

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

The variable i is defined only in this for loop.

This loop executes 6 times.

When To Use a `for` Loop?

- Yes, a `while` loop can do everything a `for` loop can do

- Programmers like it because it is concise
  - Initialization
  - Condition
  - Update
  - All on one line!

In general, the `for` loop:

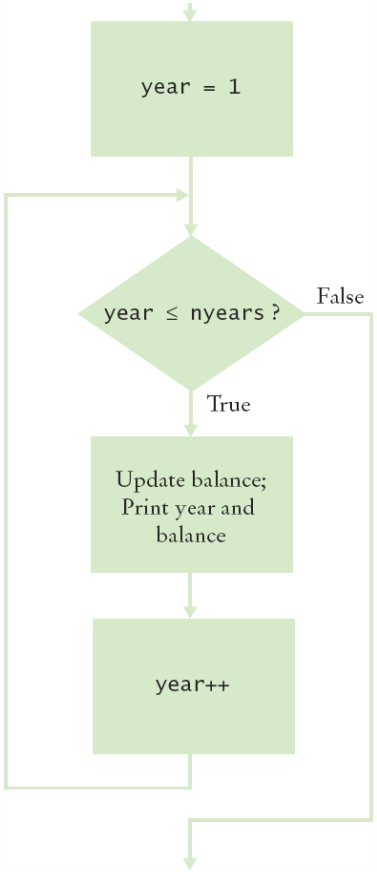
```
for (initialization; condition; update)
{
    statements
}
```

has exactly the same effect as the `while` loop:

```
initialization;
while (condition)
{
    statements
    update
}
```

Planning a for Loop

- Print the balance at the end of each year for a number of years



Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

```
for (int year = 1; year <= nyears; year++)
{
    Update balance.
    Print year and balance.
}
```

InvestmentTable.java

- Setup variables
- Get input
- Loop
  - Calc
  - Output

```java
import java.util.Scanner;

/**
   This program prints a table showing the growth of an investment.
*/
public class InvestmentTable
{
   public static void main(String[] args)
   {
      final double RATE = 5;
      final double INITIAL_BALANCE = 10000;
      double balance = INITIAL_BALANCE;

      System.out.print("Enter number of years: ");
      Scanner in = new Scanner(System.in);
      int nyears = in.nextInt();

      // Print the table of balances for each year

      for (int year = 1; year <= nyears; year++)
      {
         double interest = balance * RATE / 100;
         balance = balance + interest;
         System.out.printf("%4d %10.2f\n", year, balance);
      }
   }
}
```

Good Examples of for Loops

Table 2 for Loop Examples

Loop	Values of i	Comment
`for (i = 0; i <= 5; i++)`	0 1 2 3 4 5	Note that the loop is executed 6 times.
`for (i = 5; i >= 0; i--)`	5 4 3 2 1 0	Use `i--` for decreasing values.
`for (i = 0; i < 9; i = i + 2)`	0 2 4 6 8	Use `i = i + 2` for a step size of 2.
`for (i = 0; i != 9; i = i + 2)`	0 2 4 6 8 10 12 14 ... (infinite loop)	You can use `<` or `<=` instead of `!=` to avoid this problem.
`for (i = 1; i <= 20; i = i * 2)`	1 2 4 8 16	You can specify any rule for modifying `i`, such as doubling it in every step.
`for (i = 0; i < str.length(); i++)`	0 1 2 . . . until the last valid index of the string `str`	In the loop body, use the expression `str.charAt(i)` to get the `i`th character.

for Loop Variable Scope

- Scope is the 'lifetime' of a variable.

- When 'x' is declared in the `for` statement:

- 'x' exists only inside the 'block' of the for loop { }

```
for( int x = 1; x < 10; x = x + 1) {
   // steps to do inside the loop
   // You can use 'x' anywhere in this box
}
if (x > 100)    // Error! x is out of scope!
```

- Solution: Declare 'x' outside the `for` loop

```
int x;
for(x = 1; x < 10; x = x + 1)
```

Self Check

How many numbers does this loop print?

```
for (int n = 10; n >= 0; n--) {
   System.out.println(n);
}
```

Answer: 11 numbers: 10 9 8 7 6 5 4 3 2 1 0

Write a `for` loop that prints all even numbers between 10 and 20 (inclusive).

Answer:

```
for (int i = 10; i <= 20; i = i + 2) {
   System.out.println(i);
}
```

Write a `for` loop that computes the sum of the integers from 1 to `n`.

Answer:

```
int sum = 0;
for (int i = 1; i <= n; i++) {
   sum = sum + i;
}
```

Programming Tip

- Use `for` loops for their intended purposes only
  - Increment (or decrement) by a constant value
  - Do not update the counter inside the body
    - Update in the third section of the header

```
for (int counter = 1; counter <= 100; counter++)
{
   if (counter % 10 == 0) // Skip values divisible by 10
   {
      counter++; // Bad style: Do NOT update the counter inside loop
   }
   System.out.println(counter);
}
```

- Most counters start at one ‘end’ (0 or 1)
  - Many programmers use an integer named `i` for ‘index’ or ‘counter’ variable in `for` loops

Programming Tip

- Choose loop bounds that match your task
  - `for` loops establish lower and upper bounds on an index
  - When the same conditional operator is used for both bounds, the bounds are called **symmetric**

```
1 <= i <= 10
```

  - When different condintional operators are used for both bounds, the bounds are called **asymmetric**

```
1 <= i < 10
```

  - Both kinds of bounds can be appropriate in loops
  - Asymmetric bounds work well with strings

```
for(int i = 0; i < str.length(); i++)
```

Programming Tip

- Count Iterations
  - Many bugs are ‘off by one’ issues
  - One too many or one too few

- How many posts are there?

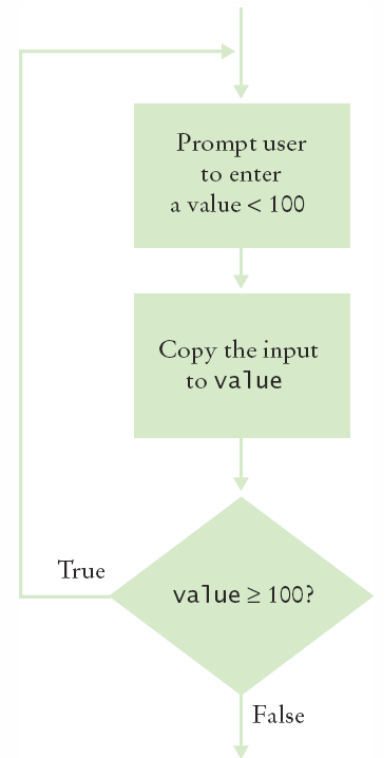
- How many pairs of rails are there?

```
final int RAILS = 5;
for (int i = 1; i < RAILS; i++ )
{
   System.out.println("Painting rail " + i);
}
```

```
Painting rail 1
Painting rail 2
Painting rail 3
Painting rail 4
```

▪The do Loop

- Use a `do` loop when you want to:
  - Execute the body at least once
  - Test the condition AFTER your first loop




```
int i = 1;  // initialize
final int FINGERS = 5;
do
{
   // paint finger
   i++; // update
}
while (i <= FINGERS);  // test
```

Note the semicolon at the end!

do Loop Example

- User Input Validation:
  - Range check a value entered
  - User must enter something to validate first!

```
int value;
do
{
   System.out.println("Enter an integer < 100: ");
   value = in.nextInt();
}
while (value >= 100);  // test
```

Self Check

Suppose that we want to check for inputs that are at least 0 and at most 100. Modify the `do` loop in this section for this check.

Answer:

```
do {
   System.out.print("Enter a value between 0 and 100: ");
   value = in.nextInt();
}while (value < 0 || value > 100);
```

Rewrite the input check `do` loop using a `while` loop. What is the disadvantage of your solution?

Answer:

```
int value = 100;
while (value >= 100){
   System.out.print("Enter a value < 100: ");
   value = in.nextInt();
}
```

Here, the variable value had to be initialized with an artificial value to ensure that the loop is entered at least once.

Self Check

Write a `do` loop that reads integers and computes their sum. Stop when reading the value 0.

Answer:

```
int x;
int sum = 0;
do {
   x = in.nextInt();
   sum = sum + x;
} while (x != 0);
```

Write a `do` loop that reads integers and computes their sum. Stop when reading a zero or the same value twice in a row. For example, if the input is 1 2 3 4 4, then the sum is 14 and the loop stops.

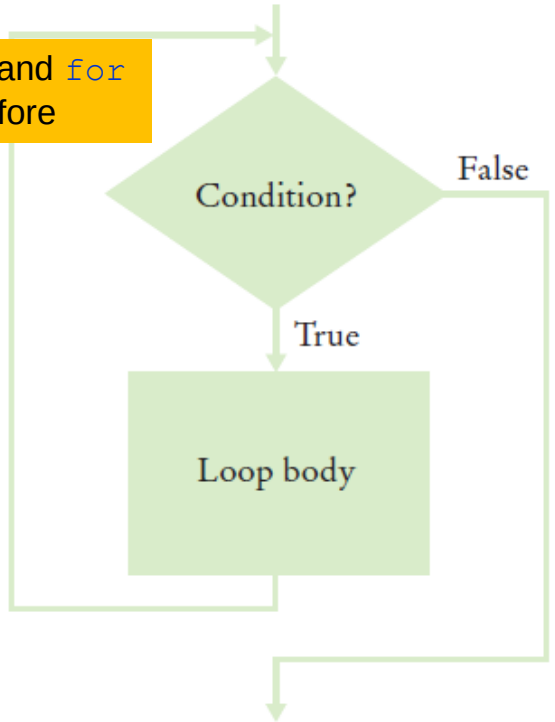
Answer:

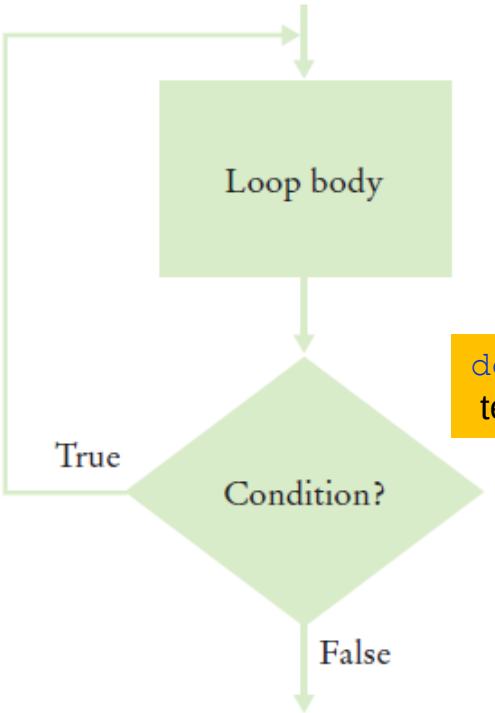
```
int x = 0;
int previous;
do {
   previous = x;
   x = in.nextInt();
   sum = sum + x;
} while (x != 0 && previous != x);
```

Programming Tip

- Flowcharts for loops
  - To avoid ‘spaghetti code’, never have an arrow that points inside the loop body

`while` and `for` test before





`do` tests after

Processing Sentinel Values

- A sentinel value denotes the end of a data set, but it is not part of the data

- Sentinel values are often used:
  - When you don't know how many items are in a list, use a 'special' character or value to signal no more items.
  - For numeric input of positive numbers, it is common to use the value -1:

```
salary = in.nextDouble();
while (salary != -1)
{
   sum = sum + salary;
   count++;
   salary = in.nextDouble();
}
```

Averaging a Set of Values

- Declare and initialize a ‘sum’ variable to 0
- Declare and initialize a ‘count’ variable to 0
- Declare and initialize an ‘input’ variable to 0
- Prompt user with instructions
- Loop until sentinel value is entered
    - Save entered value to input variable
    - If input is not -1 (sentinel value)
        - Add input to sum variable
        - Add 1 to count variable
- Make sure you have at least one entry before you divide!
    - Divide sum by count and output. Done!

SentinelDemo.java (1)

```
    public static void main(String[] args)
    {
        double sum = 0;
        int count = 0;
        double salary = 0;
        System.out.print("Enter salaries, -1 to finish: ");
        Scanner in = new Scanner(System.in);

        // Process data until the sentinel is entered

        while (salary != -1)
        {
            salary = in.nextDouble();
            if (salary != -1)
            {
                sum = sum + salary;
                count++;
            }
        }

```

Outside the `while` loop, declare and initialize variables to use

Since `salary` is initialized to 0, the `while` loop statements will be executed

Input new `salary` and compare to sentinel

Update running `sum` and `count` to average later

SentinelDemo.java (2)

```
28        // Compute and print the average
29
30        if (count > 0)
31        {
32           double average = sum / count;
33           System.out.println("Average salary: " + average);
34        }
35        else
36        {
37           System.out.println("No data");
38        }
39     }
40  }
```

Prevent divide by 0

Calculate and output the average salary using `sum` and `count` variables

Program Run

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

Boolean Variables and Sentinels

- A boolean variable can be used to control a loop
  - Sometimes called a ‘flag’ variable

```
System.out.print("Enter salaries, -1 to finish: ");
boolean done = false;
while (!done)
{
   value = in.nextDouble();
   if (value == -1)
   {
      done = true;
   }
   else
   {
      // Process value
   }
}
```

To Input Any Numeric Value…

- When valid values can be positive or negative
  - You cannot use -1 (or any other number) as a sentinel

- One solution is to use a non-numeric sentinel
  - But Scanner's `in.nextDouble` will fail!
  - Use Scanner's `in.hasNextDouble` first
    - Returns a boolean: `true` (all's well) or `false` (not a number)
    - Then use `in.nextDouble` if `true`

```
System.out.print("Enter values, Q to quit: ");
while (in.hasNextDouble())
{
   value = in.nextDouble();
   // Process value
}
```

Self Check

What is wrong with the following loop for reading a sequence of values?

```
System.out.print("Enter values, Q to quit: ");
do
{
   double value = in.nextDouble();
   sum = sum + value;
   count++;
}
while (in.hasNextDouble());
```

Answer: If the user doesn't provide any numeric input, the first call to `in.nextDouble()` will fail.

Common Loop Algorithms

1: Sum and Average Value

2: Counting Matches

3: Finding the First Match

4: Prompting until a match is found

5: Maximum and Minimum

6: Comparing Adjacent Values

Sum and Average Examples

- Sum of Values
  - Initialize total to 0
  - Use while loop with sentinel

```
double total = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
}
```

- Average of Values
  - Use Sum of Values
  - Initialize count to 0
    - Increment per input
  - Check for count 0
    - Before divide!

```
double total = 0;
int count = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
   count++;
}
double average = 0;
if (count > 0)
  { average = total / count; }
```

Counting Matches

- Counting Matches
- Initialize count to 0
- Use a `for` loop
- Add to count per match

```
int upperCaseLetters = 0;
for (int i = 0; i < str.length(); i++)
{
   char ch = str.charAt(i);
   if (Character.isUpperCase(ch))
   {
      upperCaseLetters++;
   }
}
```

Finding the First Match

- Initialize `boolean` sentinel to `false`
- Initialize position counter to 0
  - First char in String
- Use a compound conditional in loop

```
boolean found = false;
char ch;
int position = 0;
while (!found &&
       position < str.length())
{
   ch = str.charAt(position);
   if (Character.isLowerCase(ch))
   {
      found = true;
   }
   else { position++; }
}
```

Prompt Until a Match Is Found

- Initialize `boolean` flag to `false`

- Test sentinel in `while` loop
  - Get input, and compare to range
    - If input is in range, change flag to `true`
    - Loop will stop executing

```
boolean valid = false;
double input;
while (!valid)
{
   System.out.print("Please enter a positive value < 100: ");
   input = in.nextDouble();
   if (0 < input && input < 100) { valid = true; }
   else { System.out.println("Invalid input."); }
}
```

Maximum and Minimum

- Get first input value
  - This is the largest (or smallest) that you have seen so far!
  -
- Loop while you have a valid number (non-sentinel)
  - Get another input value
  - Compare new input to largest (or smallest)
  - Update largest (or smallest), if necessary

```
double largest = in.nextDouble();
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   if (input > largest)
   {
      largest = input;
   }
}
```

```
double smallest = in.nextDouble();
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   if (input > smallest)
   {
      smallest = input;
   }
}
```

Comparing Adjacent Values

- Get first input value

- Use `while` to determine if there are more to check
  - Copy input to previous variable
  - Get next value into input variable
  - Compare input to previous, and output if same

```
double input = in.nextDouble();
while (in.hasNextDouble())
{
   double previous = input;
   input = nextDouble();
   if (input == previous)
   {
      System.out.println("Duplicate input");
   }
}
```

Self Check

How do you compute the total of all positive inputs?

Answer:

```
double total = 0;
while (in.hasNextDouble()){
   double input = in.nextDouble();
   if (input > 0){
      total = total + input;
   }
}
```

What is wrong with the following loop for finding the position of the first space in a string?

```
boolean found = false;
for (int position = 0; !found && position < str.length(); position++){
   char ch = str.charAt(position);
   if (ch == ' ') { found = true; }
}
```

Answer: The loop will stop when a match is found, but you cannot access the match because neither `position` nor `ch` are defined outside the loop.

Self Check

How do you find the position of the *last* space in a string?

Answer: Start the loop at the end of string:

```
boolean found = false;
int i = str.length() - 1;
while (!found && i >= 0)
{
   char ch = str.charAt(i);
   if (ch == ' ') { found = true; }
   else { i--; }
}
```

Steps to Writing a Loop

Planning:

1. Decide what work to do inside the loop
2. Specify the loop condition
3. Determine loop type
4. Setup variables before the first loop
5. Process results when the loop is finished
6. Trace the loop with typical examples

Coding:

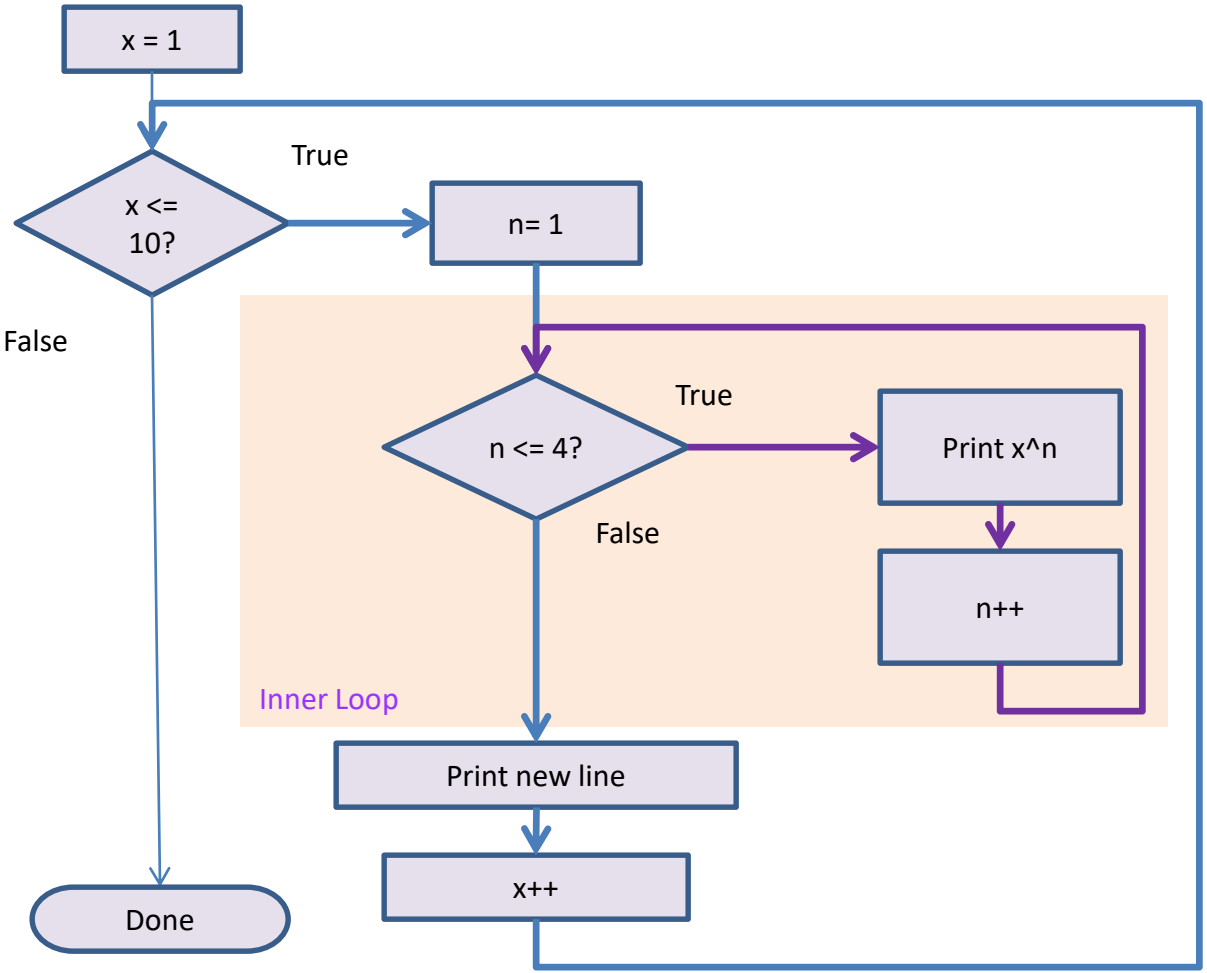
7. Implement the loop in Java

Nested Loops

- How would you print a table with rows and columns?
  - Print top line (header)
    - Use a `for` loop
  - Print table body…
    - How many rows?
    - How many columns?
  - Loop per row
    - Loop per column

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
…	…	…	…
10	100	1000	10000

Flowchart of a Nested Loop



PowerTable.java

```java
/**
   This program prints a table of powers of x.
*/
public class PowerTable
{
   public static void main(String[] args)
   {
      final int NMAX = 4;
      final double XMAX = 10;

      // Print table body

      for (double x = 1; x <= XMAX; x++)
      {
         // Print table row

         for (int n = 1; n <= NMAX; n++)
         {
            System.out.printf("%10.0f", Math.pow(x, n));
         }
         System.out.println();
      }
   }
}
```

Body of outer loop

Body of inner loop

```
         1         2         3         4
         x         x         x         x

         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```

Nested Loop Examples (1)

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 4; j++)  { print "*" }`
`  print new line`
`}`
	****
	****
	****
		Prints 3 rows of 4 asterisks each.

`for (i = 1; i <= 4; i++)`
`{`
`  for (j = 1; j <= 3; j++) { print "*" }`
`  print new line`
`}`
	***
	***
	***
	***
		Prints 4 rows of 3 asterisks each.

`for (i = 1; i <= 4; i++)`
`{`
`  for (j = 1; j <= i; j++) { print "*" }`
`  print new line`
`}`
	*
	**
	***
	****
		Prints 4 rows of lengths 1, 2, 3, and 4.

Nested Loop Examples (2)

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 5; j++)`
`  {`
`  if (j % 2 == 0) { Print "*" }`
`  else { Print "-" }`
`  }`
`  System.out.println();`
`}`
	-*-*-
	-*-*-
	-*-*-
		Prints asterisks in even columns, dashes in odd columns.

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 5; j++)`
`  {`
`  if (i % 2 == j % 2) { Print "*" }`
`  else { Print " " }`
`  }`
`  System.out.println();`
`}`
	* * *
	 * *
	* * *
		Prints a checkerboard pattern.

Self Check

What do the following nested loops display?

```
for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 4; j++) {
      System.out.print(i + j);
   }
   System.out.println();
}
```

Answer:

```
0123
1234
2345
```

Write nested loops that make the following pattern of brackets:

```
[][][][]
[][][][]
[][][][]
```

Answer:

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 4; j++){
        System.out.print("[]");
    }
    System.out.println();
}
```

Random Numbers and Simulations

- Games often use random numbers to make things interesting
  - Rolling Dice
  - Spinning a wheel
  - Pick a card

- A simulation usually involves looping through a sequence of events
  - Days
  - Events

RandomDemo.java

```java
/**
   This program prints ten random numbers between 0 and 1.
*/
public class RandomDemo
{
   public static void main(String[] args)
   {
      for (int i = 1; i <= 10; i++)
      {
         double r = Math.random();
         System.out.println(r);
      }
   }
}
```

Program Run

```
0.2992436267816825
0.43860176045313537
0.7365753471168408
0.6880250194282326
0.1608272403783395
0.5362876579988844
0.3098705906424375
0.6602909916554179
0.1927951611482942
0.8632330736331089
```

Simulating Die Tosses

- Goal
  - Get a random integer between 1 and 6

```java
/**
   This program simulates tosses of a pair of dice.
*/
public class Dice
{
   public static void main(String[] args)
   {
      for (int i = 1; i <= 10; i++)
      {
         // Generate two random numbers between 1 and 6

         int d1 = (int) (Math.random() * 6) + 1;
         int d2 = (int) (Math.random() * 6) + 1;
         System.out.println(d1 + " " + d2);
      }
      System.out.println();
   }
}
```

Program Run

```
5 1
2 1
1 2
5 1
1 2
6 4
4 4
6 1
6 3
5 2
```

Self Check

How do you generate a random floating-point number ≥ 0 and < 100?

Answer: `Math.random() * 100.0`