Advanced Computer Programming

[Lecture 04]

Saeed Reza Kheradpisheh

kheradpisheh@ut.ac.ir

Department of Computer Science
Shahid Beheshti University

Spring 1397-98

LOOPS

In a loop, a part of a program is **repeated** over and over, until a specific goal is reached. Loops are important for calculations that require repeated steps and for processing input consisting of many data items.

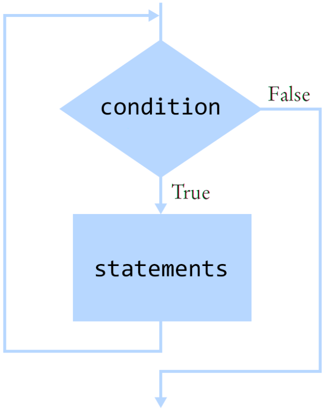
The while Loop

Usage

Executing instructions repeatedly
while a condition is true.

Syntax

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



Localization

Definition

When you declare a variable inside the loop body, the variable becomes local to the loop, which is created for each iteration and removed after the end of each iteration.

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
} // interest no longer declared here
```

A new `interest` variable is created in each iteration.

Localization

Definition

When you declare a variable inside the loop body, the variable becomes local to the loop, which is created for each iteration and removed after the end of each iteration.

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
} // interest no longer declared here
```

A new interest variable is created in each iteration.

In contrast, for variables that were declared outside the loop body, the same variable is used for all iterations of the loop.

Infinite Loops

Definition

An **infinite loop** is a loop that <u>runs forever</u> and can be stopped only by killing the program or restarting the computer.

- A common reason for infinite loops is forgetting to update the variable that controls the loop.
- Another common reason for an infinite loop is accidentally incrementing a counter that should be decremented (or vice versa).
- Press Ctrl+C or close the program's window to kill a program containing infinite loop.

The `while` Loop: Examples

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" (see Common Error 4.2 on page 145).
`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.
`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 (see Common Error 4.1 on page 144).
`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.

Hand-Tracing

Definition

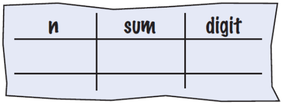
Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.

Hand-Tracing

Definition

Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.

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



n	sum	digit

Hand-Tracing

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

n	sum	digit
1729	0	

Exercise (`Star.java`)

Write a program that takes `n` and prints the pattern bellow:

```
*
**
***
****
*****
...
```

where there should be exactly `n` lines of stars.

The `for` Loop

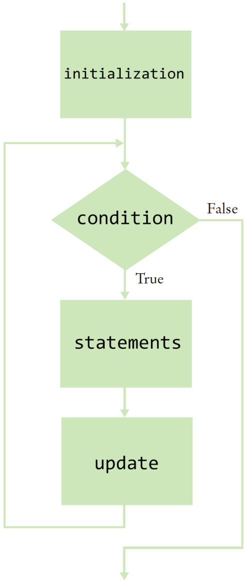
Usage

The `for` loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.

Syntax

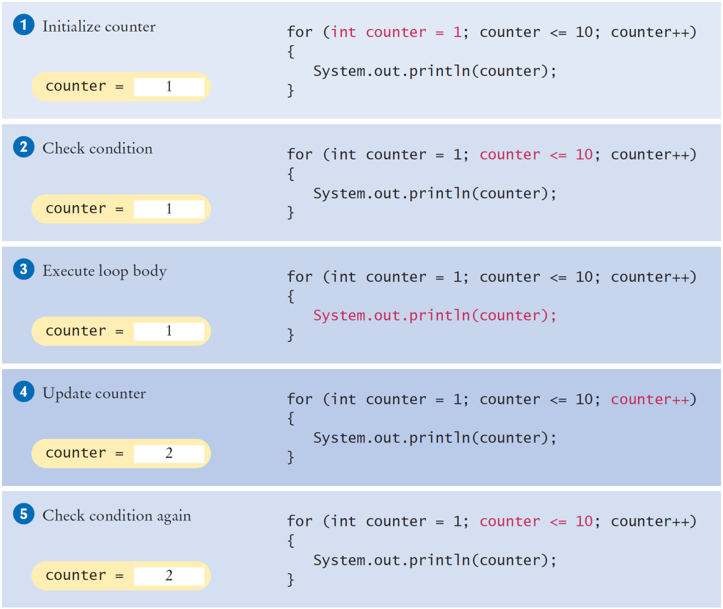
```
for(initialization; condition; update)
{
    //body
    statements
}
```

The `for` Loop



- The `initialization` is executed once, before the loop is entered.
- The `condition` is checked before each iteration.
- The `update` is executed after each iteration.

The `for` Loop: Example



```
for (int counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

```
for (int counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

```
for (int counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

```
for (int counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

```
for (int counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

The `for` Loop: Example

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. (See Programming Tip 4.3 on page 156.)
`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.

The `for` Loop: Example

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. (See Programming Tip 4.3 on page 156.)
`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.

The compiler won't check whether the initialization, condition, and update expressions are related

Localization in `for` Loop

- Declaring a variable inside the loop body.
  (Same as the `while` loop)
- Declaring a variable in `initialization` part.

```
for (int counter = 1; counter <= 10; counter++)
{
    . . .
}
// counter no longer declared here
```

Exercise (`Factorial.java`)

Write a program that takes `n` and computes `n!`.

The `do` Loop

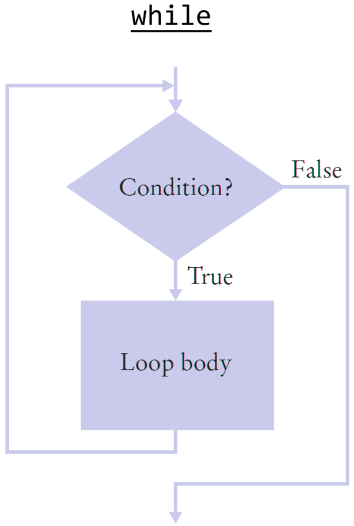
Usage

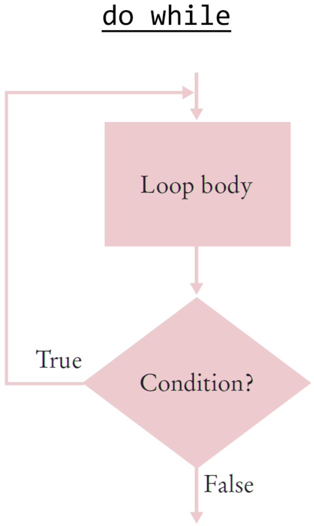
The `do` loop is appropriate when the loop body must be executed at least once.

Syntax

```
do
{
   //body
   statements
}
while (condition);
```

`while` V.S. `do while`





The Loop-and-a-Half

Consider the following loop for processing inputs until:

```
boolean done = false;
while (!done)
{
   double value = in.nextDouble();
   if (value == -1)
   {
      done = true;
   }
   else
   {
      Process value.
   }
}
```

The Loop-and-a-Half

Consider the following loop for processing inputs until:

```
boolean done = false;
while (!done)
{
   double value = in.nextDouble();
   if (value == -1)
   {
      done = true;
   }
   else
   {
      Process value.
   }
}
```

Definition

The actual test for **loop termination is in the middle of the loop**, not at the top. This is called a **loop and a half** because one must go halfway into the loop before knowing whether one needs to terminate.

Exercise (`InRange.java`)

Write a program that takes two integers `a` and `b`. Then it keeps taking integers until the user enters an integer number in range `[a,b)`.

Exercise (`Fibo.java`)

Write a program that takes `n` and computes the `n`th Fibonacci number.

Exercise (`StrRev.java`)

Write a program that takes a string `str` and prints the reverse of `str`.

Exercise (`Palindrome.java`)

Write a program that takes a string `str` and checks if it is a palindrome.

`break` Statement

Definition

The **`break` statement** breaks out of the enclosing loop, independent of the loop condition.

- When the break statement is encountered, the loop is terminated, and the statement following the loop is executed.

```
int p = 1;
while (true)
{
   double value = in.nextDouble();
   if (value == -1)
   {
      break;
   }
   p *= value;
}
```

Computing the product of input integers until -1 is entered.

`continue` Statement

Definition

The **`continue` statement** bypasses the rest of the current iteration and starts executing the next iteration.

```
int p = 1;
while (true)
{
   double value = in.nextDouble();
   if (value == -1)
   {
      break;
   }
   if (value == 0)
   {
      continue;
   }
   p *= value;
}
```

Avoiding 0 in computations.

Redirection of Input and Output

Usage

Use **input redirection** to read input from a file. Use **output redirection** to print program output in a file.

- Reading input from a file,
```
java ClassName < inputFileName
```
- Writing output to a file,
```
java ClassName > outputFileName
```
- Both reading and writing,
```
java ClassName < inputFileName > outputFileName
```

Common Loop Algorithms

- Sum and Average Value: To compute an average, keep a total and a count of all values.
- Counting Matches: To count values that fulfill a condition, check all values and increment a counter for each match.
- Finding the First Match: If your goal is to find a match, exit the loop when the match is found.
- Maximum and Minimum: To find the largest value, update the largest value seen so far whenever you see a larger one.
- Comparing Adjacent Values: To compare adjacent inputs, store the preceding input in a variable.

Nested Loops

Definition

When the body of a loop contains another loop, the loops are **nested**.

Nested Loops	Output	Explanation

`for (i = 1; i <= 3; i++)`
`{`
`   for (j = 1; j <= 4; j++)  { Print "*" }`
`   System.out.println();`
`}`
	`****`
	`****`
	`****`
		Prints 3 rows of 4 asterisks each.

`for (i = 1; i <= 4; i++)`
`{`
`   for (j = 1; j <= 3; j++) { Print "*" }`
`   System.out.println();`
`}`
	`***`
	`***`
	`***`
	`***`
		Prints 4 rows of 3 asterisks each.

`for (i = 1; i <= 4; i++)`
`{`
`   for (j = 1; j <= i; j++) { Print "*" }`
`   System.out.println();`
`}`
	`*`
	`**`
	`***`
	`****`
		Prints 4 rows of lengths 1, 2, 3, and 4.

Exercise (`Distinct.java`)

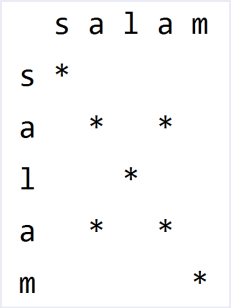
Write a program that computes the minimum number of characters that must be removed from the string `str` to have a new string with no identical adjacent characters.

Exercise (`Guess.java`)

Write a program that asks the user to guess a target integer number (the target number is hard-coded into the program). It should guide the user with ”Your guess is too low” or ”Your guess is too high” sentences.

Exercise (DotPlot.java)

Write a program that takes two strings str1 and str2 and computes the dot-plot of comparing these strings.



```
  s a l a m
s *
a   *   *
l     *
a   *   *
m         *
```

Random Numbers

- The Java library has a random number generator, which produces numbers that appear to be completely random.
- Calling `Math.random()` yields a random floating-point number that is `>= 0` and `< 1`.
- Call `Math.random()` again, and you get a different number.

```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);
      }
   }
}
```

Exercise (`Candy.java`)

Randomness is not always stressful (remember rolling your names)! Sometimes, it can be delicious like winning a candy!
Write a program that takes the number of students following by their names (from a file using input redirection), and choose a student randomly as the winner.