Computer Programming

Loops Statements

Özgür Koray ŞAHİNGÖZ
Prof.Dr.

Biruni University
Computer Engineering Department

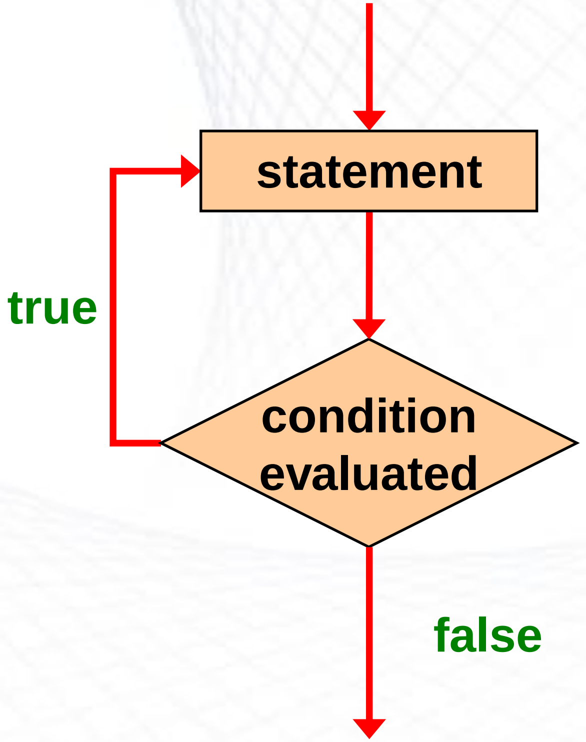
The do Statement

- A *do statement* has the following syntax:

```
do {
   statement-list;
}
while (condition);
```

- The `statement-list` is executed once initially, and then the `condition` is evaluated
- The statement is executed repeatedly until the condition becomes false

Logic of a do Loop



The do Statement

- An example of a `do` loop:

```
int count = 0;
do {
   count++;
   System.out.println(count);
} while (count < 5);
```

- The body of a `do` loop executes at least once

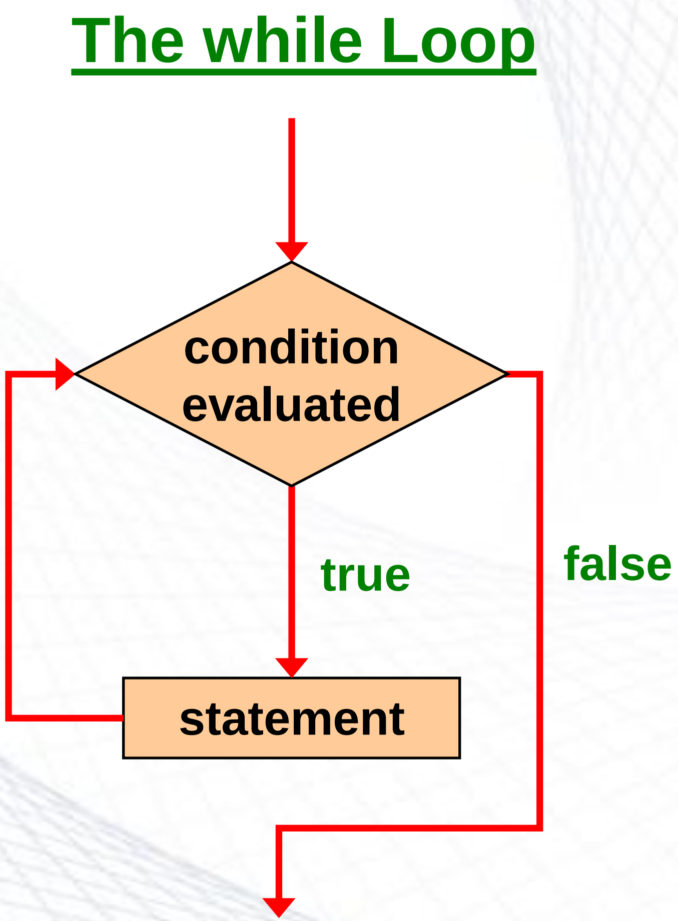
ReverseNumber.java

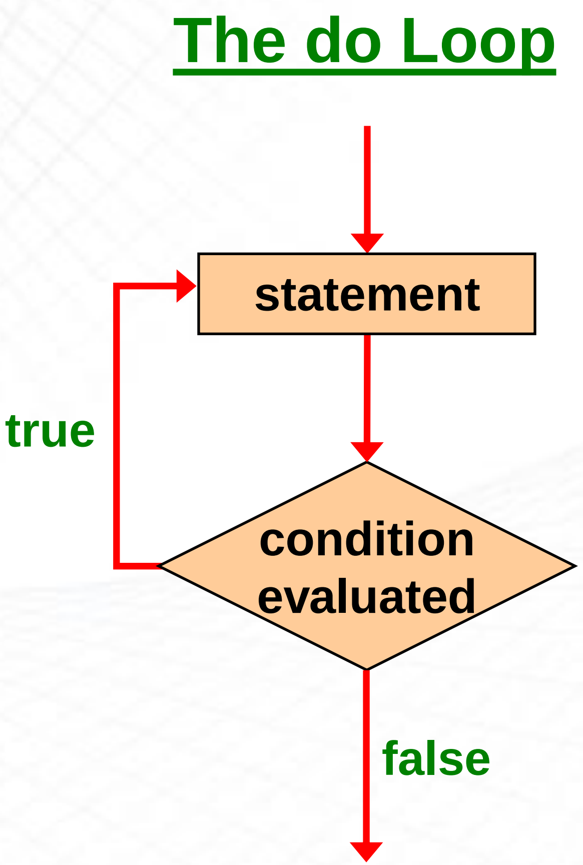
```java
import java.util.Scanner;

public class ReverseNumber {
   public static void main(String[] args)    {
      int number, lastDigit, reverse = 0;

      Scanner scan = new Scanner(System.in);
      System.out.print("Enter a positive integer: ");
      number = scan.nextInt();

      do  {
         lastDigit = number % 10;
       System.out.println(lastDigit);
         reverse = (reverse * 10) + lastDigit;
         number = number / 10;
      } while (number > 0);

      System.out.println("That number reversed is " + reverse);
   }
}
```

Sample Run

```
Enter a positive integer: 2896
6
9
8
2
That number reversed is 6982
```

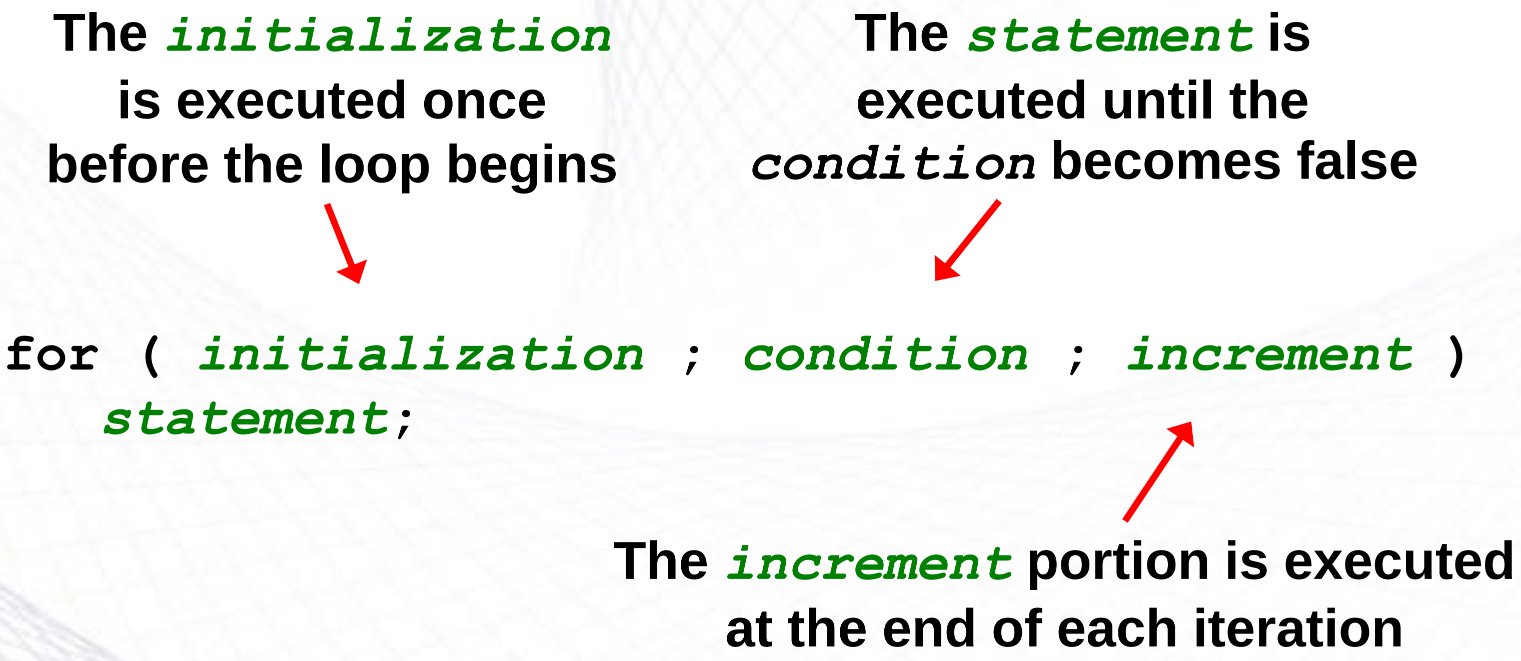
Comparing while and do





The for Statement

- A *for statement* has the following syntax:



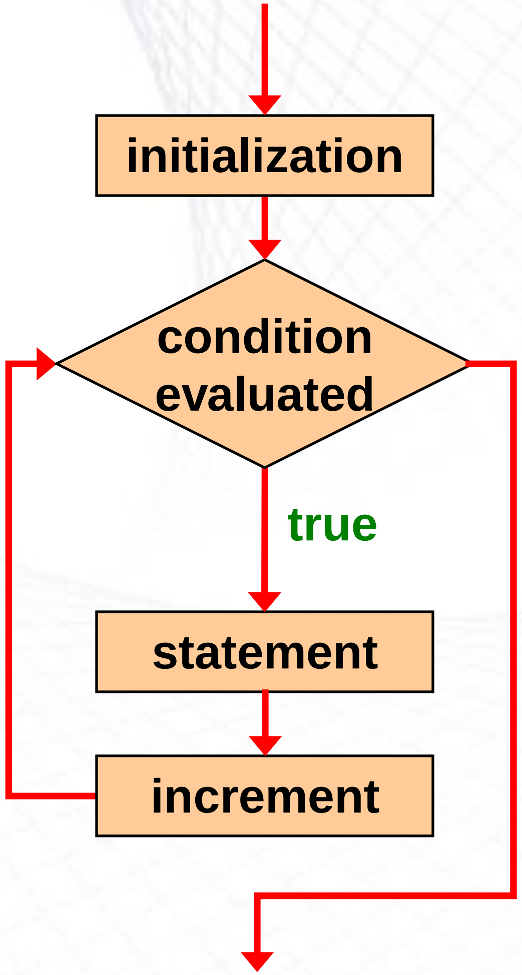
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

Logic of a for loop



The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition ){
    statement;
    increment;
}
```

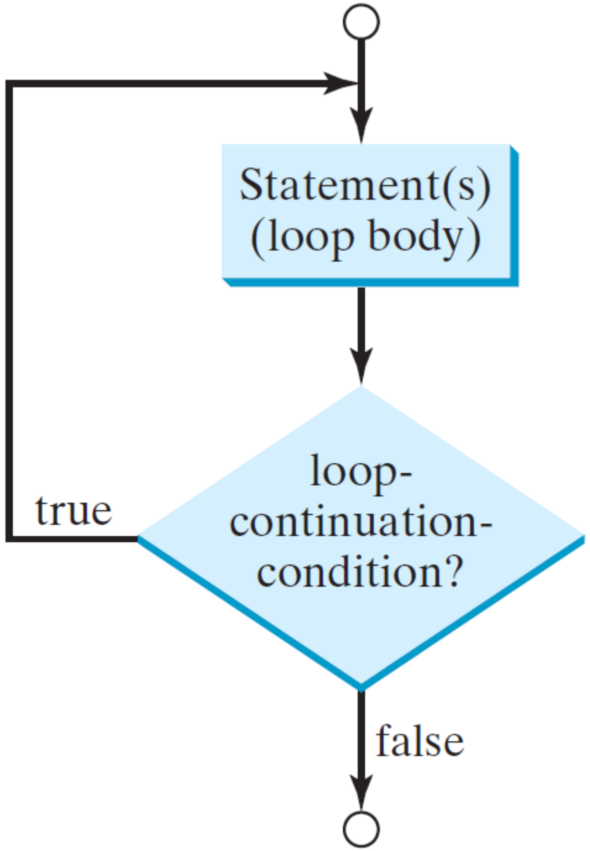
The for Statement

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
    System.out.println(count);
```

- The initialization section can be used to declare a variable
- Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times

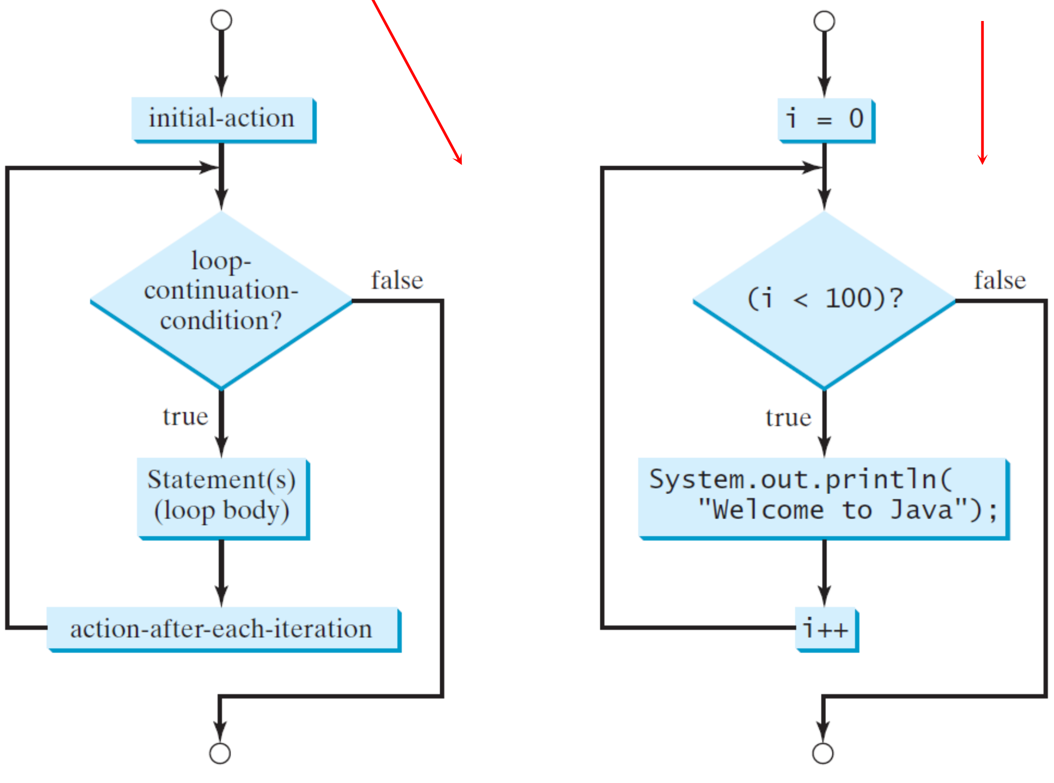
do-while Loop



```
do {
   // Loop body;
   Statement(s);
} while (loop-continuation-condition);
```

for Loops

```
for (initial-action; loop-continuation-condition; action-after-each-iteration) {
   // loop body;
   Statement(s);
}
```



(a)

(b)

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
     "Welcome to Java!");
}
```

Trace for Loop

Declare i

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute initializer
i is now 0

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

(i < 2) is true
since i is 0

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 1

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

**(i < 2) is still true
since i is 1**

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 2

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

(i < 2) is false
since i is 2

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Exit the loop. Execute the next statement after the loop

The for Statement

- The increment section can perform any calculation:

```
for (int num=100; num > 0; num -= 5)
   System.out.println(num);
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance
- See `Stars.java`
- See `Multiples.java`

Stars.java

```
public class Stars {
   //-----------------------------------------------------------------
   //  Prints 10 * to  the screen.
   //-----------------------------------------------------------------
   public static void main(String[] args){
      final int MAX_ROWS = 10;

      for (int star = 1; star <= MAX_ROWS; star++)
            System.out.print("*");

         System.out.println();
      }
}
```

Output

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

Stars.java

```
public class Stars {
   //-----------------------------------------------------------------
   //  Prints a triangle shape using asterisk (star) characters.
   //-----------------------------------------------------------------
   public static void main(String[] args){
      final int MAX_ROWS = 10;

      for (int row = 1; row <= MAX_ROWS; row++){
         for (int star = 1; star <= row; star++)
            System.out.print("*");

         System.out.println();
      }
   }
}
```

Output

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

Multiples

```java
import java.util.Scanner;

public class Multiples {
  public static void main(String[] args)   {
      final int PER_LINE = 5; int value, limit, mult, count = 0;

      Scanner scan = new Scanner(System.in);

      System.out.print("Enter a positive value:  ");
      value = scan.nextInt();
      System.out.print("Enter an upper limit:  ");
      limit = scan.nextInt();

     System.out.println("The multiples of " + value + " between "+value + " and " + limit + " are:");

      for (mult = value; mult <= limit; mult += value) {
         System.out.print(mult + "\t");
         count++;
         if (count % PER_LINE == 0)
            System.out.println();
      }
   }
}
```

Sample Run

```
Enter a positive value: 7
Enter an upper limit: 400

The multiples of 7 between 7 and 400 are:
7       14      21      28      35
42      49      56      63      70
77      84      91      98      105
112     119     126     133     140
147     154     161     168     175
182     189     196     203     210
217     224     231     238     245
252     259     266     273     280
287     294     301     308     315
322     329     336     343     350
357     364     371     378     385
392     399
```

Note

- The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for (int i = 1; i < 100; System.out.println(i++));


for (int i = 0, j = 0; (i + j < 10); i++, j++) {
  // Do something
}
```

Note

- If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in
  - (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in
  - (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

Logic Error

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

Caution, cont.

Similarly, the following loop is also wrong:

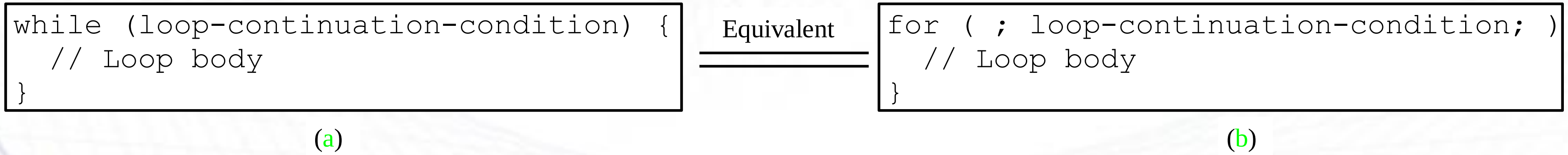
```
int i=0;
while (i < 10);     ←  Logic Error
{
  System.out.println("i is " + i);
  i++;
}
```

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
  System.out.println("i is " + i);
  i++;
} while (i<10);     ←  Correct
```

Which Loop to Use?

- The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):



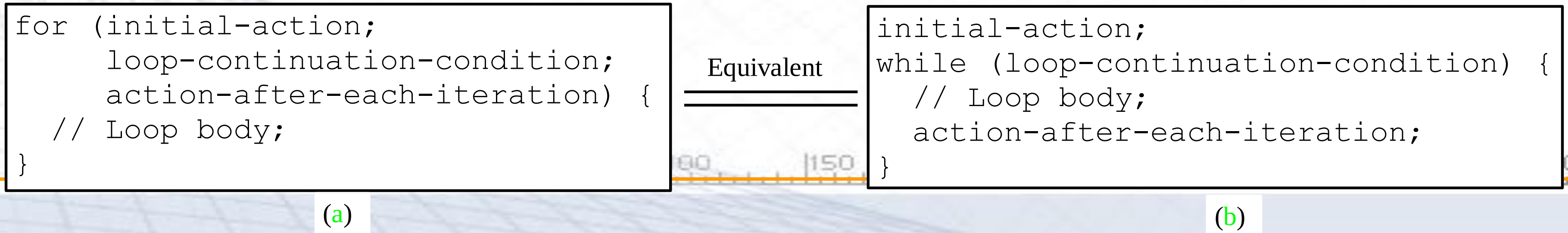
```
while (loop-continuation-condition) {
  // Loop body
}
```

(a)

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(b)

- A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):



```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

(a)

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

(b)

Recommendations

- Use the one that is most intuitive and comfortable for you. In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times. A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Nested Loops

Problem: Write a program that uses nested for loops to print a multiplication table.

MultiplicationTable

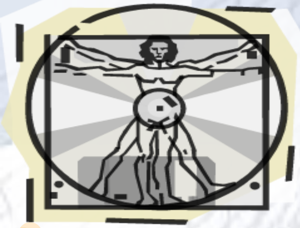
Run

Minimizing Numerical Errors

Numeric errors involving floating-point numbers are inevitable. This section discusses how to minimize such errors through an example.

Here is an example that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

TestSum Run



Problem:
Finding the Greatest Common Divisor

- Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.
- Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

GreatestCommonDivisor **Run**

Problem: Predicting the Future Tuition

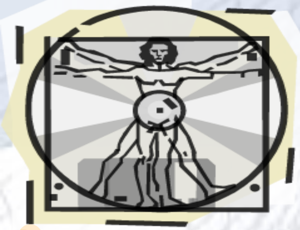
- Problem: Suppose that the tuition for a university is $10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

FutureTuition

Run

Problem: Predicating the Future Tuition

- double tuition = 10000;   int year = 0 // Year 0
- tuition = tuition * 1.07; year++;   // Year 1
- tuition = tuition * 1.07; year++;   // Year 2
- tuition = tuition * 1.07; year++;   // Year 3
- ...



Case Study: *Converting Decimals to Hexadecimals*

Hexadecimals are often used in computer systems programming (see Appendix F for an introduction to number systems). How do you convert a decimal number to a hexadecimal number? To convert a decimal number *d* to a hexadecimal number is to find the hexadecimal digits h_n, h_{n-1}, h_{n-2}, ... , h_2, h_1, and h_0 such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \ldots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

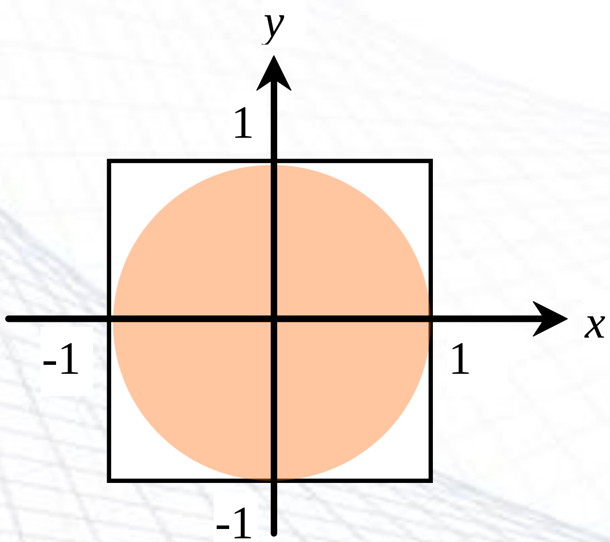
These hexadecimal digits can be found by successively dividing *d* by 16 until the quotient is 0. The remainders are h_0, h_1, h_2, ... , h_{n-2}, h_{n-1}, and h_n.

Dec2Hex **Run**

Companion Website

Problem: *Monte Carlo Simulation*

- The Monte Carlo simulation refers to a technique that uses random numbers and probability to solve problems. This method has a wide range of applications in computational mathematics, physics, chemistry, and finance. This section gives an example of using the Monto Carlo simulation for estimating π.



circleArea / squareArea = π / 4.

π can be approximated as 4 * numberOfHits / numberOfTrials

MonteCarloSimulation **Run**

Using break and continue

Examples for using the break and continue keywords:

- TestBreak.java

TestBreak Run

- TestContinue.java

TestContinue Run

break

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

  while (number < 20) {
   number++;
   sum += number;
   if (sum >= 100)
    break;
  }

  System.out.println("The number is " + number);
  System.out.println("The sum is " + sum);
 }
}
```

continue

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

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
```

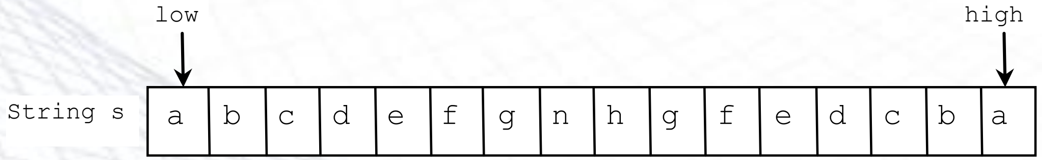
Guessing Number Problem Revisited

Here is a program for guessing a number. You can rewrite it using a break statement.

GuessNumberUsingBreak Run

Problem: Checking Palindrome

- A string is a palindrome if it reads the same forward and backward. The words “mom,” “dad,” and “noon,” for instance, are all palindromes.
- The problem is to write a program that prompts the user to enter a string and reports whether the string is a palindrome. One solution is to check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-to-last character. This process continues until a mismatch is found or all the characters in the string are checked, except for the middle character if the string has an odd number of characters.



Palindrome

Run

Problem: Displaying Prime Numbers

- Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

- Solution: The problem can be broken into the following tasks:
  - For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
  - Determine whether a given number is prime.
  - Count the prime numbers.
  - Print each prime number, and print 10 numbers per line.

PrimeNumber **Run**