7. Repetition Statements

17 Sep 2015

Objectives

- Implement repetition control in a program using ‘while’ statements.
- Implement repetition control in a program using ‘do–while’ statements.
- Implement repetition control in a program using ‘for’ statements.

Objectives

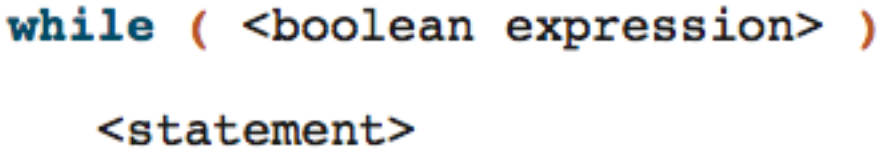
- Nest a loop repetition statement inside another repetition statement.
- Choose the appropriate repetition control statement for a given task.
- Format output values by using the Formatter class.
- Write simple recursive methods.

Repetition Statements

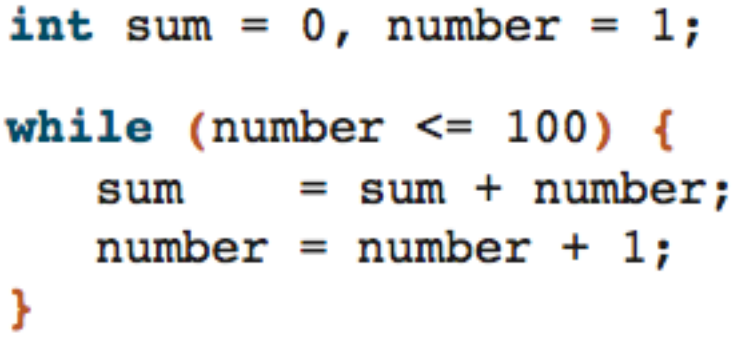
- Repetition statements control a block of code to be executed for a fixed number of times or until a certain condition is met. We will describe three repetition statements:
  1. while
  2. do–while
  3. for

The while Statement

- The while statement is a ‘pretest loop’ follows the general format



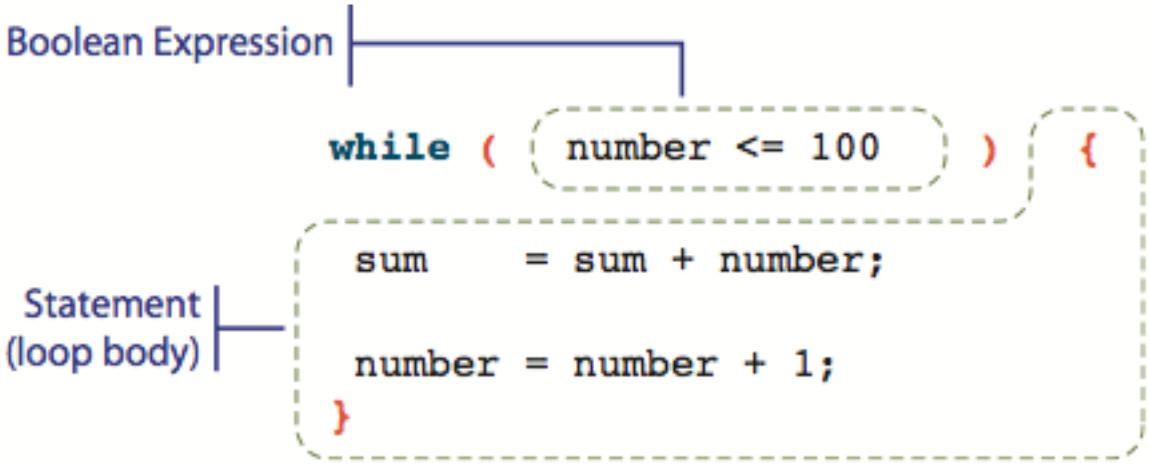
```
while ( <boolean expression> )

   <statement>
```



```
int sum = 0, number = 1;

while (number <= 100) {
    sum    = sum + number;
    number = number + 1;
}
```

The while Statement

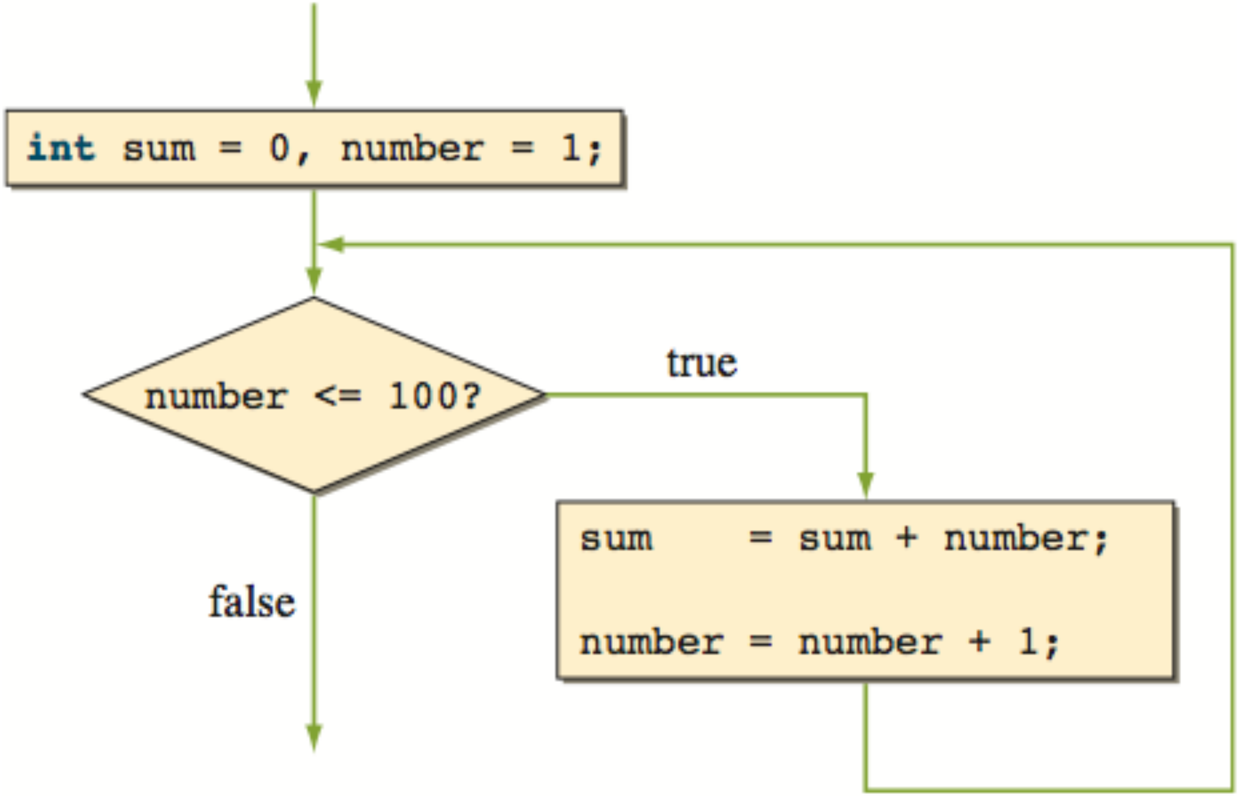
- The while statement corresponds to the general format.



```
while ( number <= 100 ) {
  sum    = sum + number;

  number = number + 1;
}
```

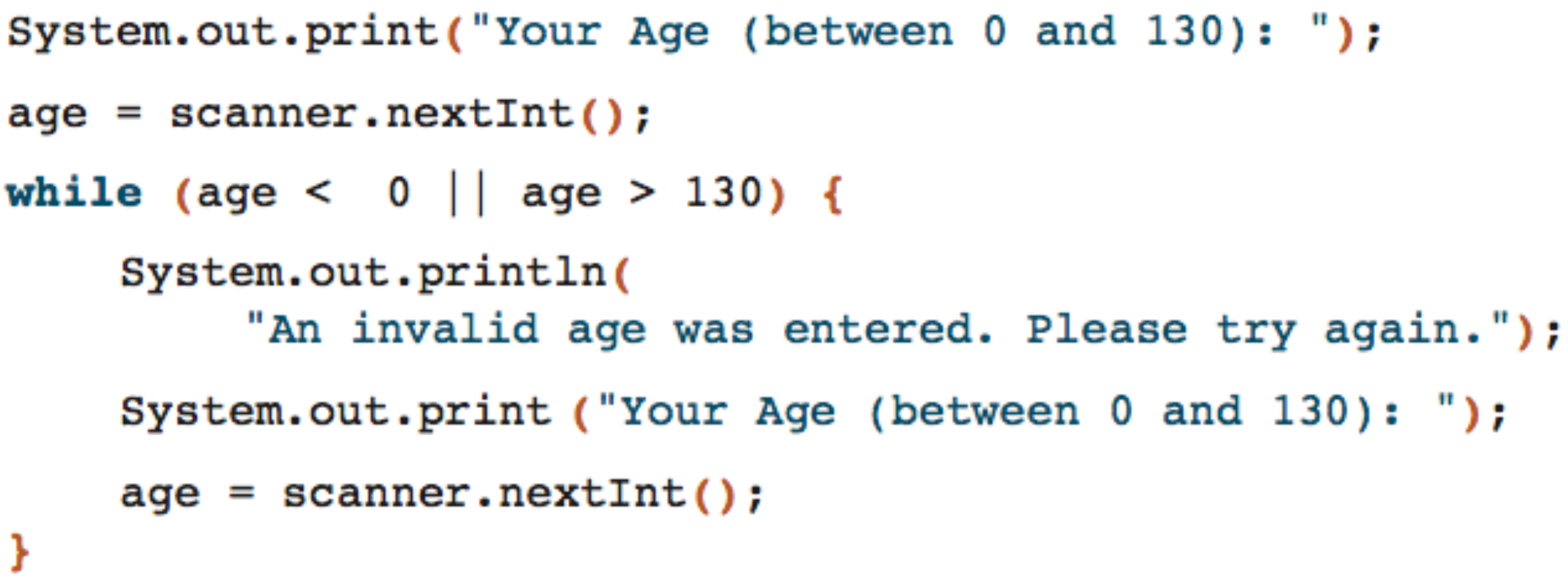
The while Statement

- A diagram showing the control flow of a while statement.



The while Statement

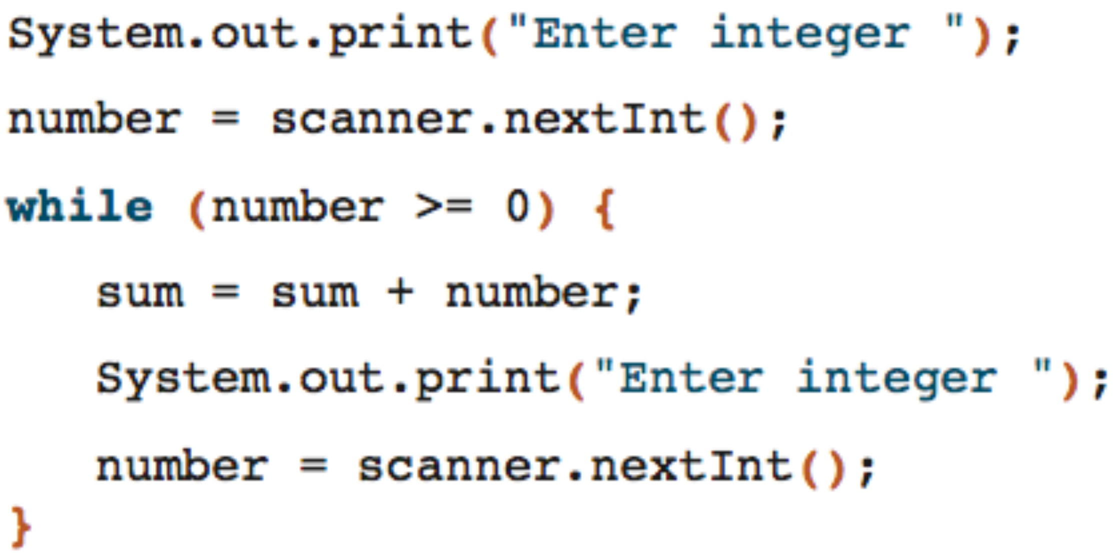
- Ex. We can use while loop to improve user interface like this example.



```
System.out.print("Your Age (between 0 and 130): ");
age = scanner.nextInt();
while (age <  0 || age > 130) {
    System.out.println(
        "An invalid age was entered. Please try again.");
    System.out.print ("Your Age (between 0 and 130): ");
    age = scanner.nextInt();
}
```

The while Statement

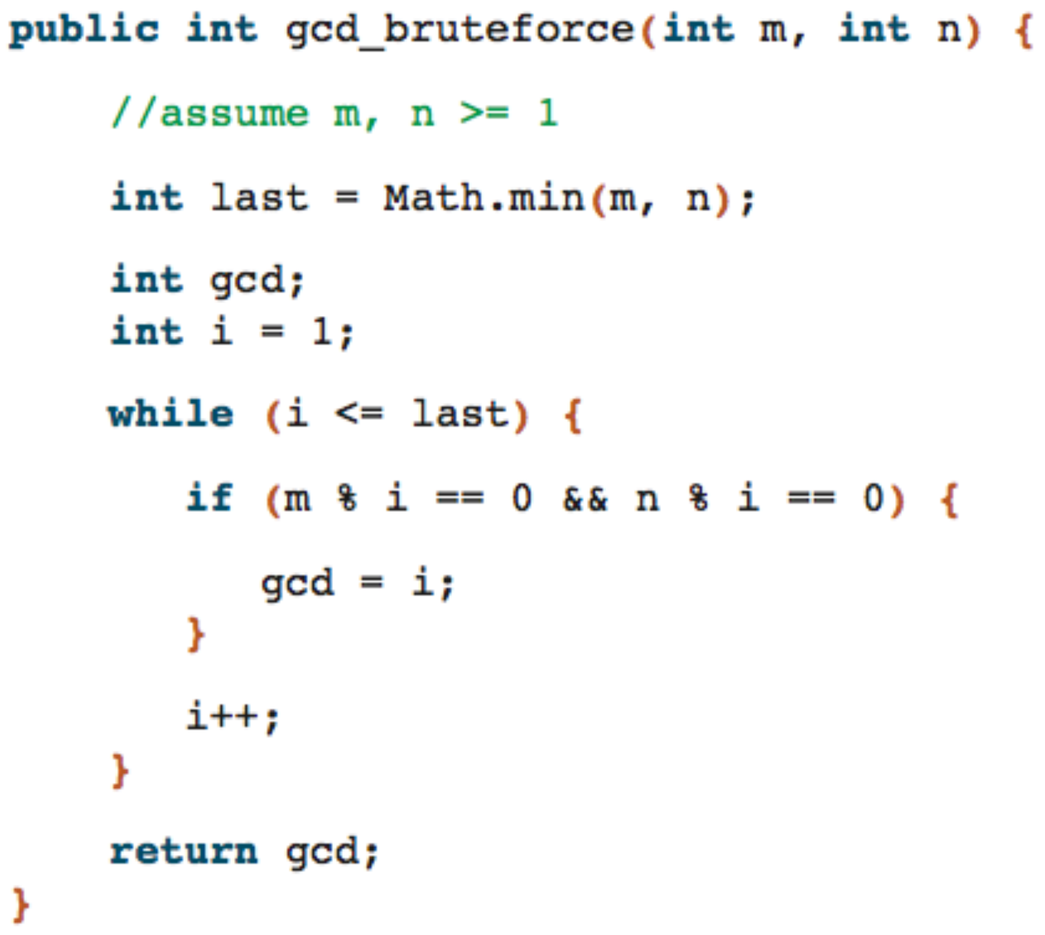
- Ex. We can use input condition to break while loop like this example.



```
System.out.print("Enter integer ");
number = scanner.nextInt();
while (number >= 0) {
    sum = sum + number;
    System.out.print("Enter integer ");
    number = scanner.nextInt();
}
```

The while Statement

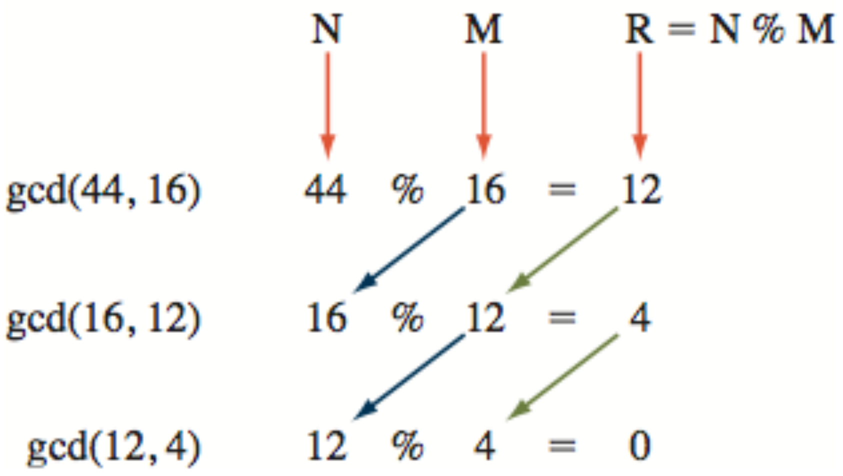
- Ex. Calculate gdc (หรม.) 1 - brute-force



```
public int gcd_bruteforce(int m, int n) {
    //assume m, n >= 1
    int last = Math.min(m, n);
    int gcd;
    int i = 1;
    while (i <= last) {
        if (m % i == 0 && n % i == 0) {
            gcd = i;
        }
        i++;
    }
    return gcd;
}
```

The while Statement

- Ex. Calculate gdc (หรม.) 2 - Euclidean algorithm



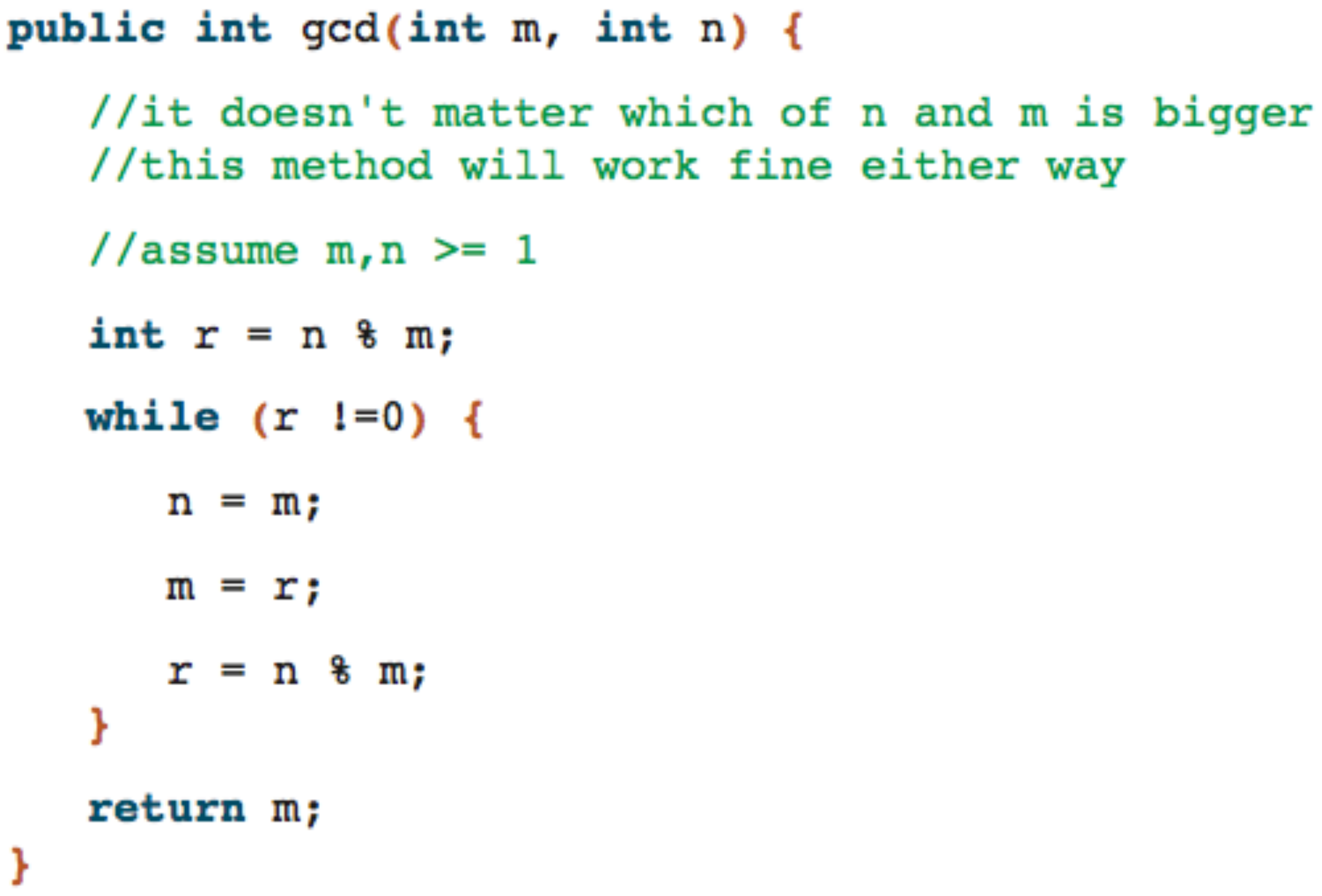
N M R = N % M

gcd(44, 16) 44 % 16 = 12

gcd(16, 12) 16 % 12 = 4

gcd(12, 4) 12 % 4 = 0

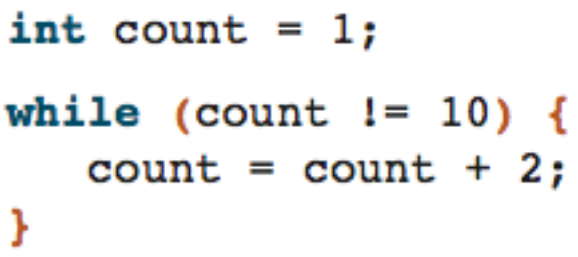
The while Statement



```
public int gcd(int m, int n) {

   //it doesn't matter which of n and m is bigger
   //this method will work fine either way

   //assume m,n >= 1

   int r = n % m;

   while (r !=0) {

      n = m;

      m = r;

      r = n % m;
   }

   return m;
}
```

Pitfalls in Repetition Statements

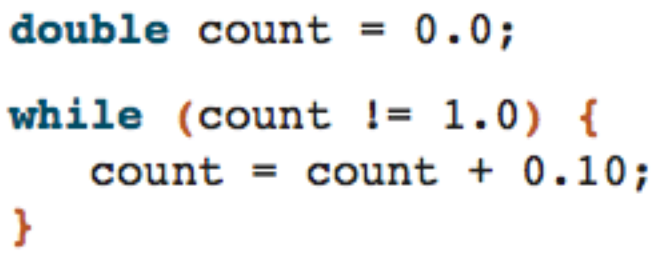
- Infinite loop: such as this one



```
int count = 1;

while (count != 10) {
   count = count + 2;
}
```

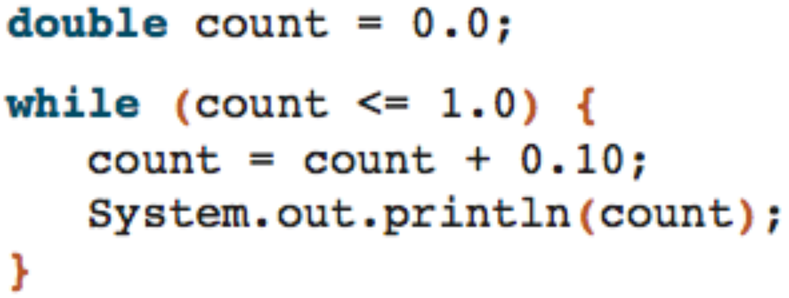
Pitfalls in Repetition Statements

- Imprecise loop counter: such as this one

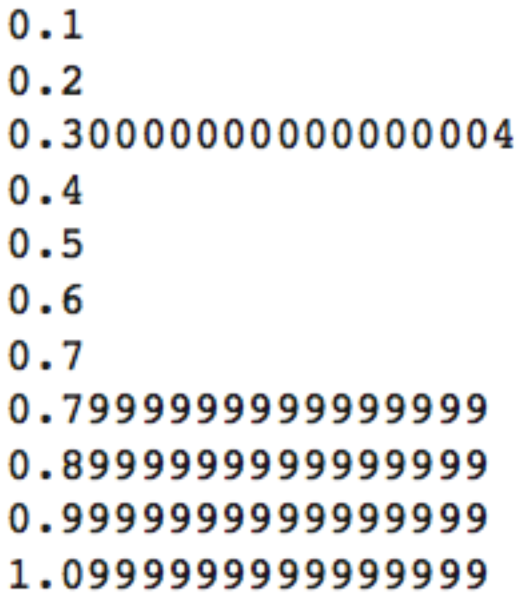


```
double count = 0.0;

while (count != 1.0) {
    count = count + 0.10;
}
```

Pitfalls in Repetition Statements



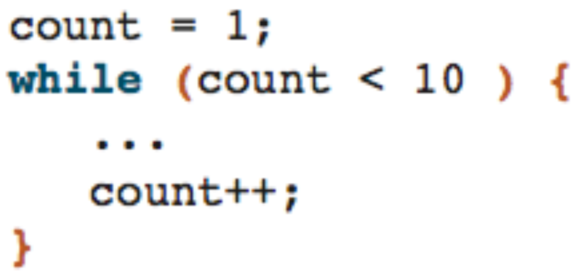
```java
double count = 0.0;

while (count <= 1.0) {
   count = count + 0.10;
   System.out.println(count);
}
```



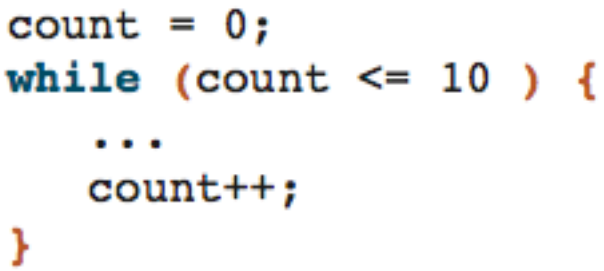
```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
```

Pitfalls in Repetition Statements

- Error of loop count
  - Suppose we want to execute the loop body 10 times (all loops below are not correct)



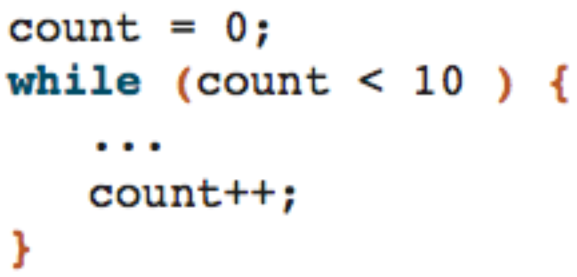
```
count = 1;
while (count < 10 ) {
    ...
    count++;
}
```



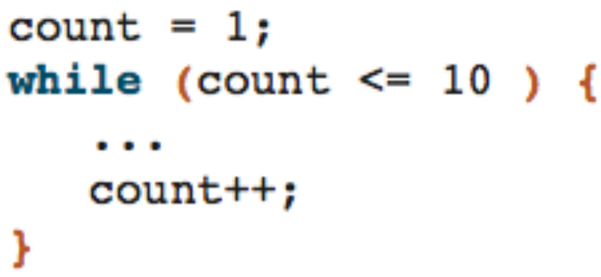
```
count = 0;
while (count <= 10 ) {
    ...
    count++;
}
```

Pitfalls in Repetition Statements

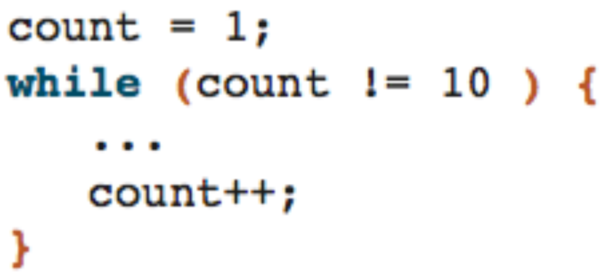
– The correct while loop are



```
count = 0;
while (count < 10 ) {
   ...
   count++;
}
```



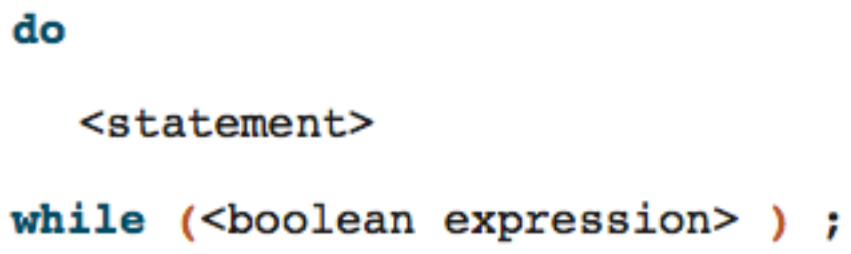
```
count = 1;
while (count <= 10 ) {
   ...
   count++;
}
```



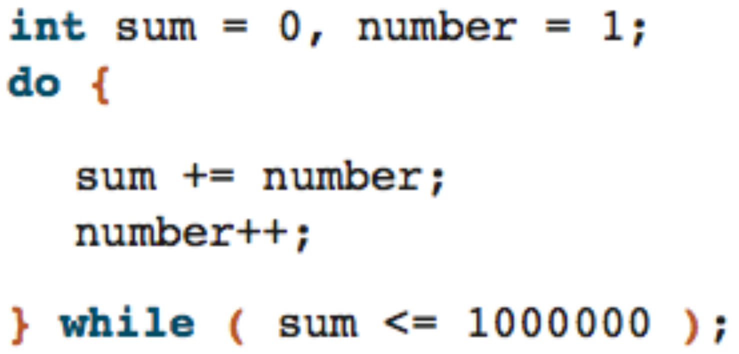
```
count = 1;
while (count != 10 ) {
   ...
   count++;
}
```

The do–while Statement

- The do-while statement is a ‘posttest loop’ follows the general format



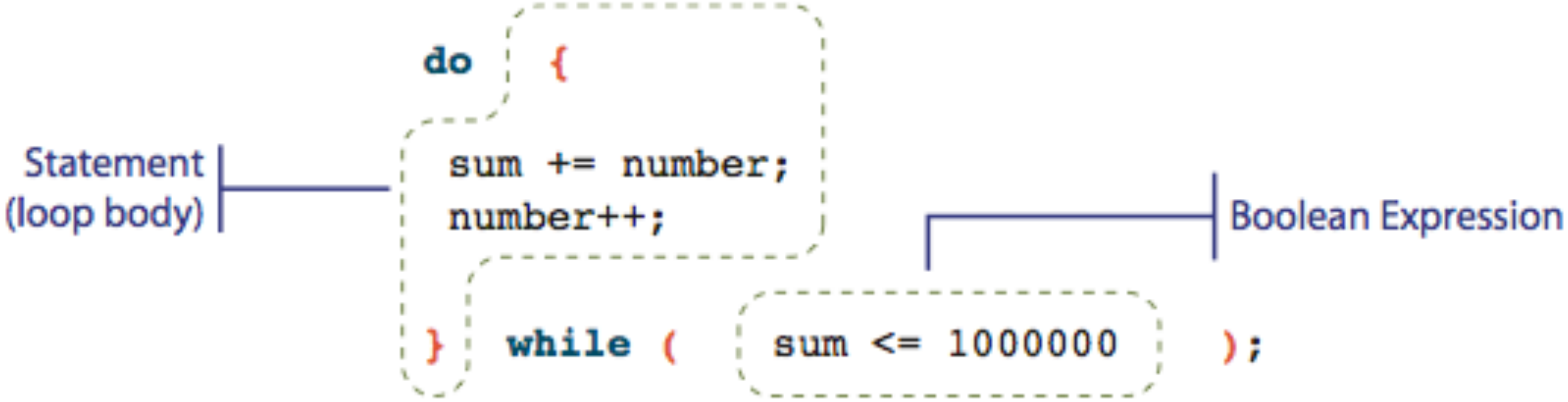
```
do
   <statement>
while (<boolean expression> ) ;
```



```
int sum = 0, number = 1;
do {
   sum += number;
   number++;
} while ( sum <= 1000000 );
```

The do–while Statement

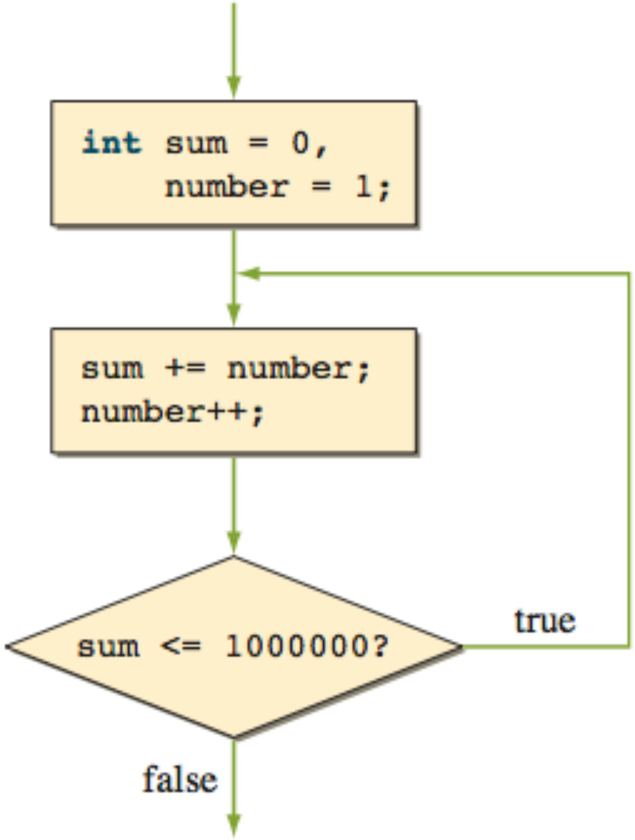
- The do-while statement corresponds to the general format.



```
do {
  sum += number;
  number++;
} while ( sum <= 1000000 );
```

The do–while Statement

- A diagram showing the control flow of a do-while statement.



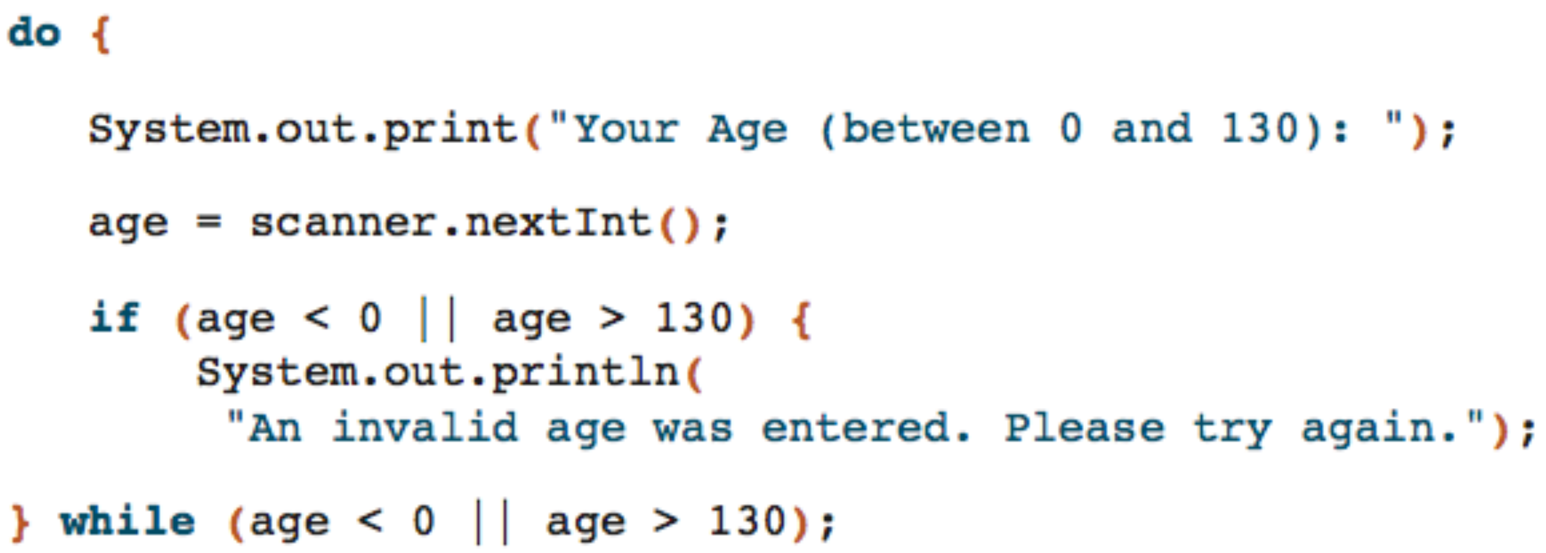
```
int sum = 0,
    number = 1;
```

```
sum += number;
number++;
```

```
sum <= 1000000?
```

The do–while Statement

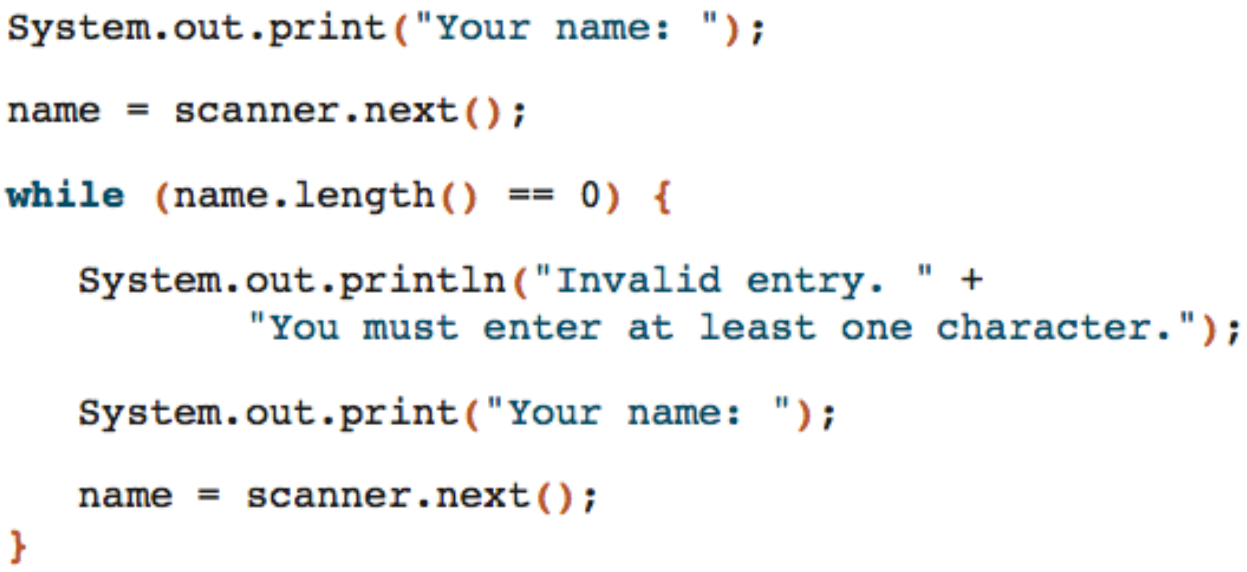
- Ex. We can use do-while loop to improve user interface like this example.



```
do {

   System.out.print("Your Age (between 0 and 130): ");

   age = scanner.nextInt();

   if (age < 0 || age > 130) {
        System.out.println(
         "An invalid age was entered. Please try again.");

} while (age < 0 || age > 130);
```

Loop-and-a-Half Repetition Control

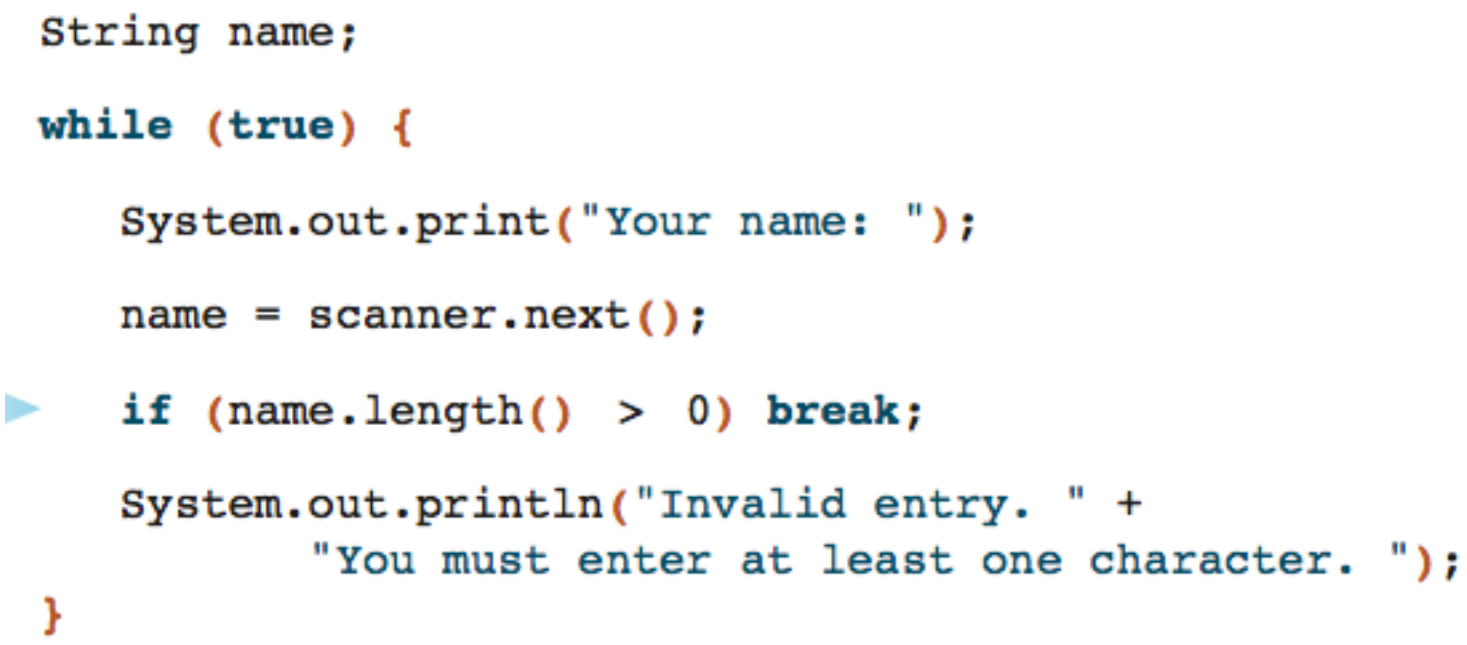
- We can test the terminating condition in the middle of the loop body, such repetition control called ‘loop-and-a-half control’.
  - Consider the following while loop



```
System.out.print("Your name: ");

name = scanner.next();

while (name.length() == 0) {

   System.out.println("Invalid entry. " +
          "You must enter at least one character.");

   System.out.print("Your name: ");

   name = scanner.next();
}
```

Loop-and-a-Half Repetition Control

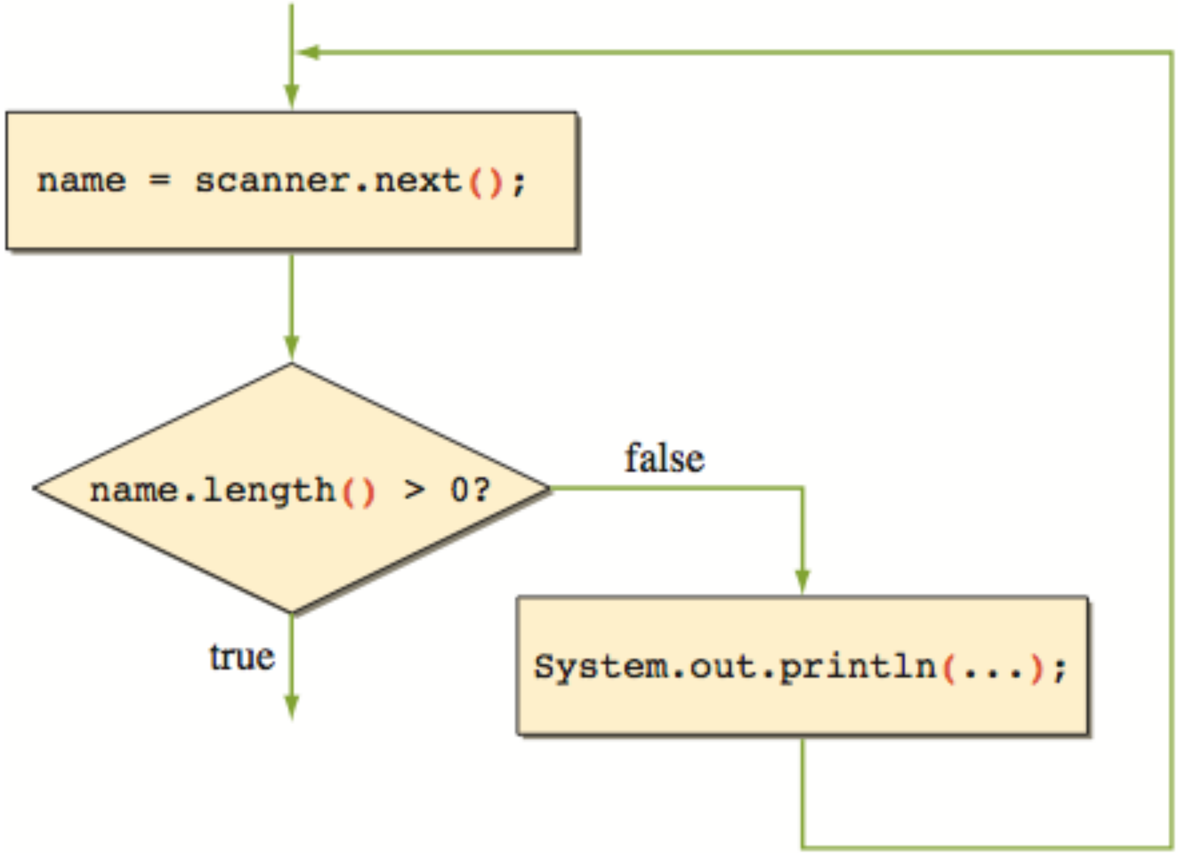
– We can avoid the duplication of code with the loop-and-a-half structure.



```
String name;
while (true) {
   System.out.print("Your name: ");
   name = scanner.next();
   if (name.length() > 0) break;
   System.out.println("Invalid entry. " +
          "You must enter at least one character. ");
}
```

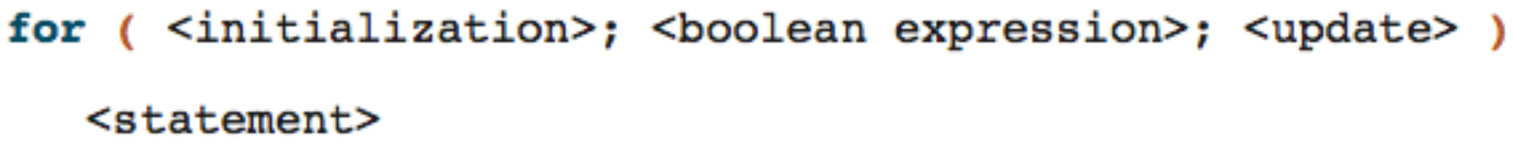
Loop-and-a-Half Repetition Control

- A diagram showing the control flow of a loop-and-a-half statement.

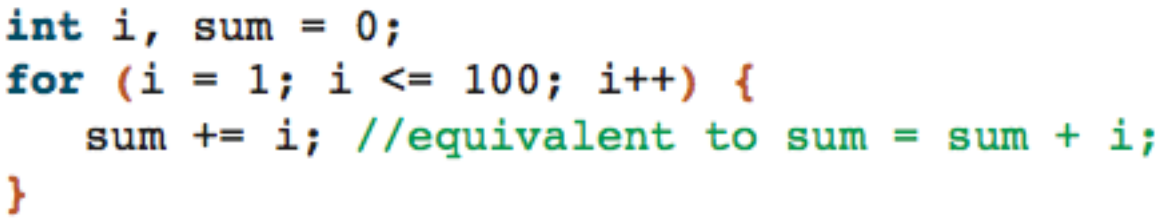


The for Statement

- The general format of the for statement is



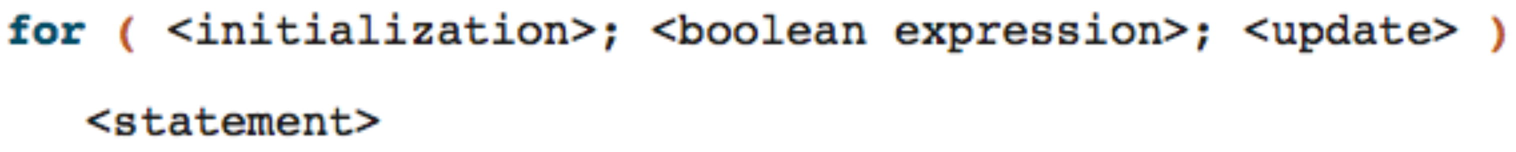
```
for ( <initialization>; <boolean expression>; <update> )
   <statement>
```



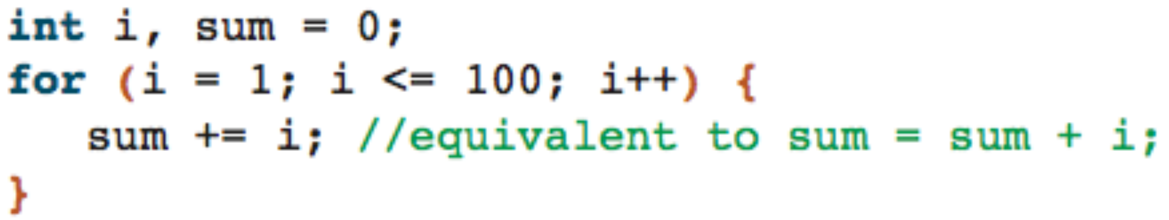
```
int i, sum = 0;
for (i = 1; i <= 100; i++) {
   sum += i; //equivalent to sum = sum + i;
}
```

The for Statement

- The general format of the for statement is



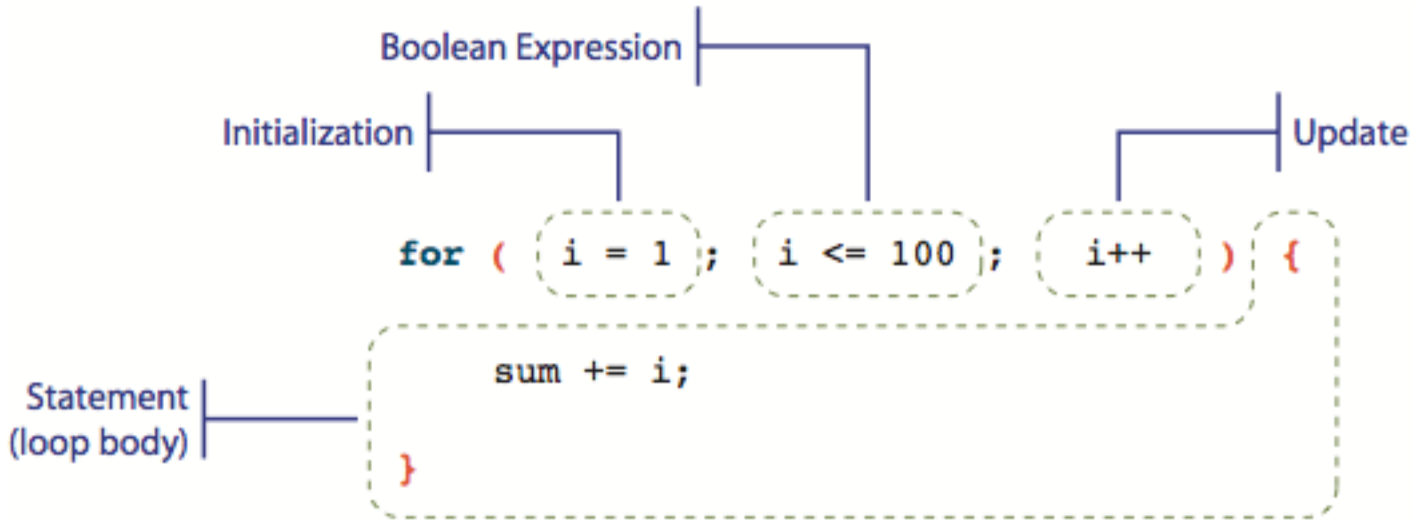
```
for ( <initialization>; <boolean expression>; <update> )
   <statement>
```



```
int i, sum = 0;
for (i = 1; i <= 100; i++) {
   sum += i; //equivalent to sum = sum + i;
}
```

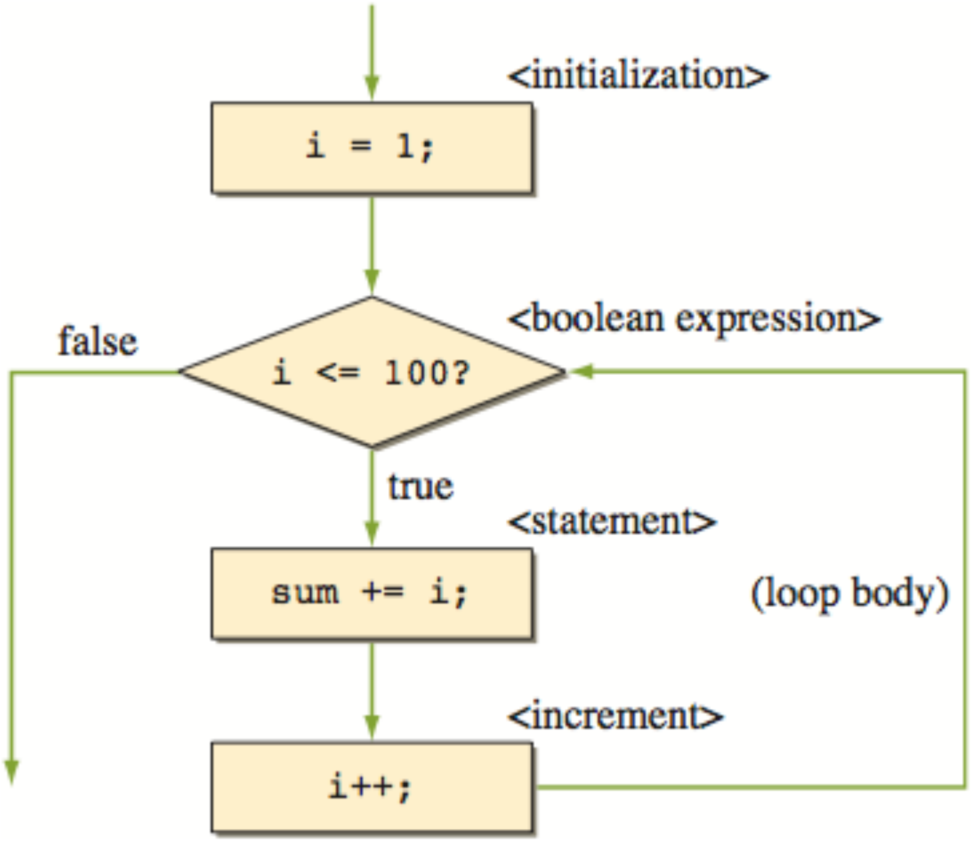
The for Statement

- The for statement corresponds to the general format.



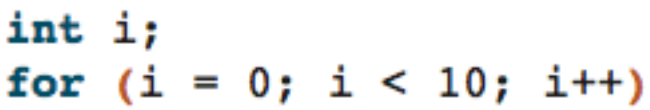
The for Statement

- A diagram showing the control flow of a for statement.

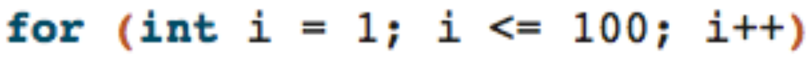


The for Statement

- The <initialization> component also can include a declaration of the control variable.



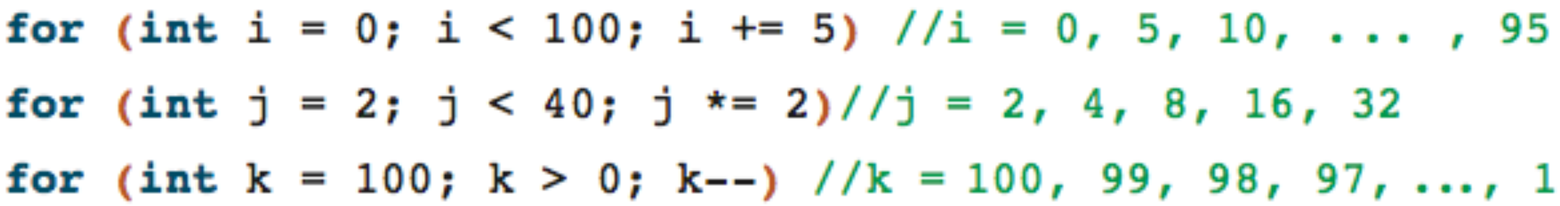
```
int i;
for (i = 0; i < 10; i++)
```



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

The for Statement

- The <update> expression in the example increments the control variable by 1. We can increment it with values other than 1, including negative values, for example:



```
for (int i = 0; i < 100; i += 5) //i = 0, 5, 10, . . . , 95
for (int j = 2; j < 40; j *= 2)//j = 2, 4, 8, 16, 32
for (int k = 100; k > 0; k--) //k = 100, 99, 98, 97, ..., 1
```

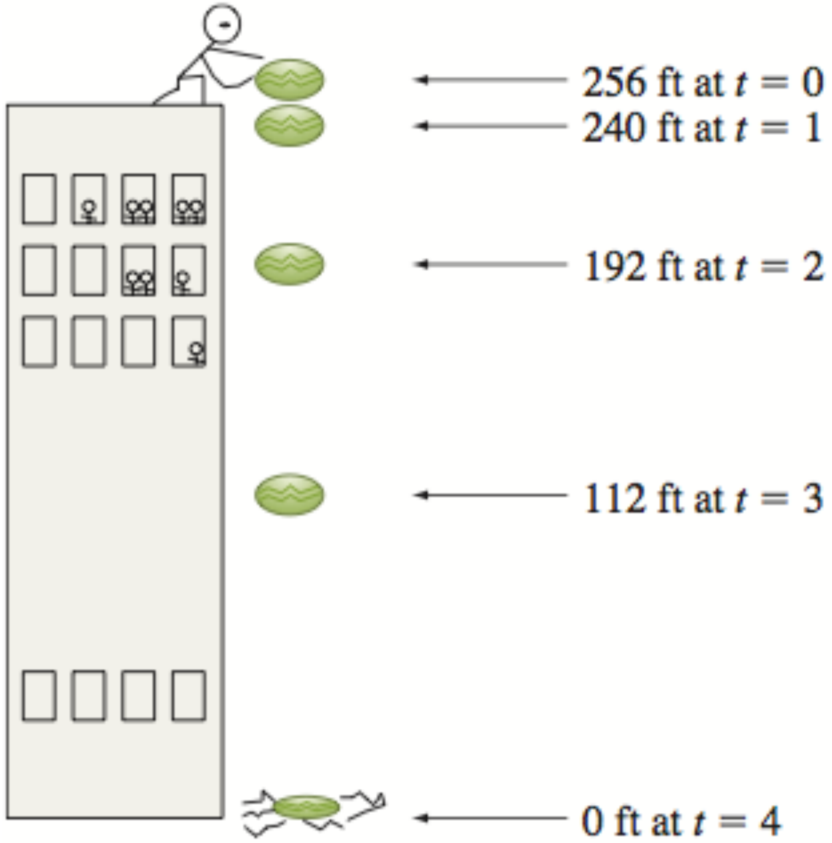
The for Statement

- Ex. Let's look at an example from physics. When an object is dropped from height H, the position P of the object at time t can be determined by the formula

$$P = -16t^2 + H$$

The for Statement

– If a watermelon is dropped from the roof of a 256-ft-high dormitory, it will drop like this:



The for Statement

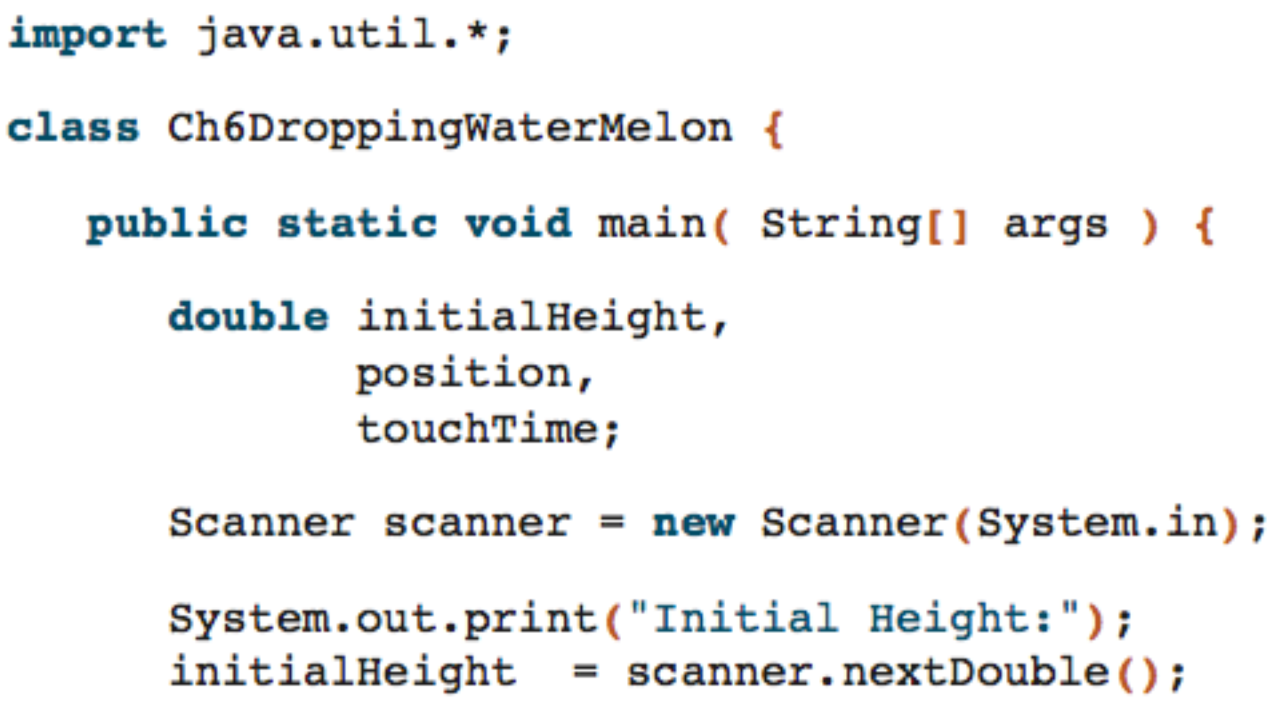
– The time the watermelon touches the ground is derived by solving for t when P = 0.

$$P = -16t^2 + H$$

$$0 = -16t^2 + H$$

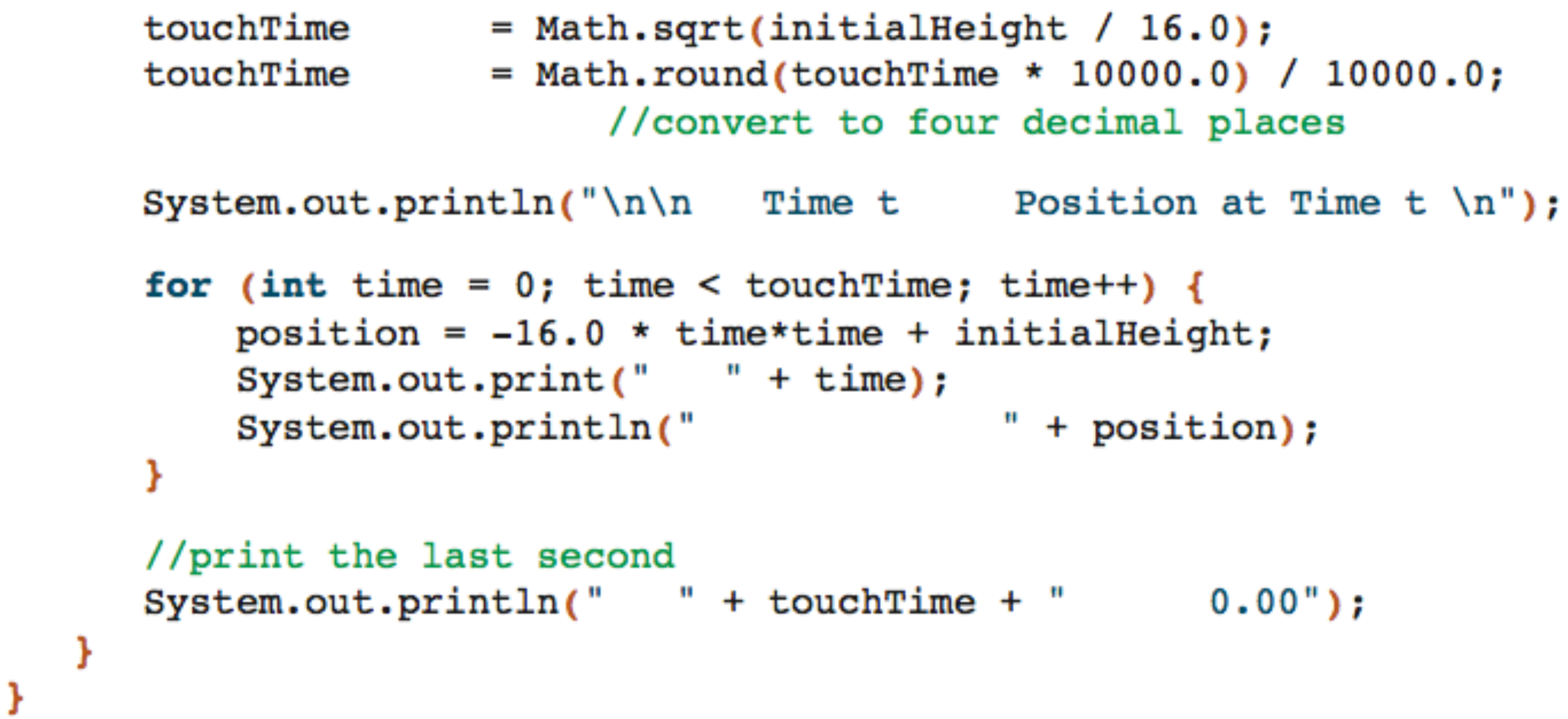
$$t = \sqrt{\frac{H}{16}}$$

The for Statement



```
import java.util.*;

class Ch6DroppingWaterMelon {

   public static void main( String[] args ) {

      double initialHeight,
             position,
             touchTime;

      Scanner scanner = new Scanner(System.in);

      System.out.print("Initial Height:");
      initialHeight  = scanner.nextDouble();
```

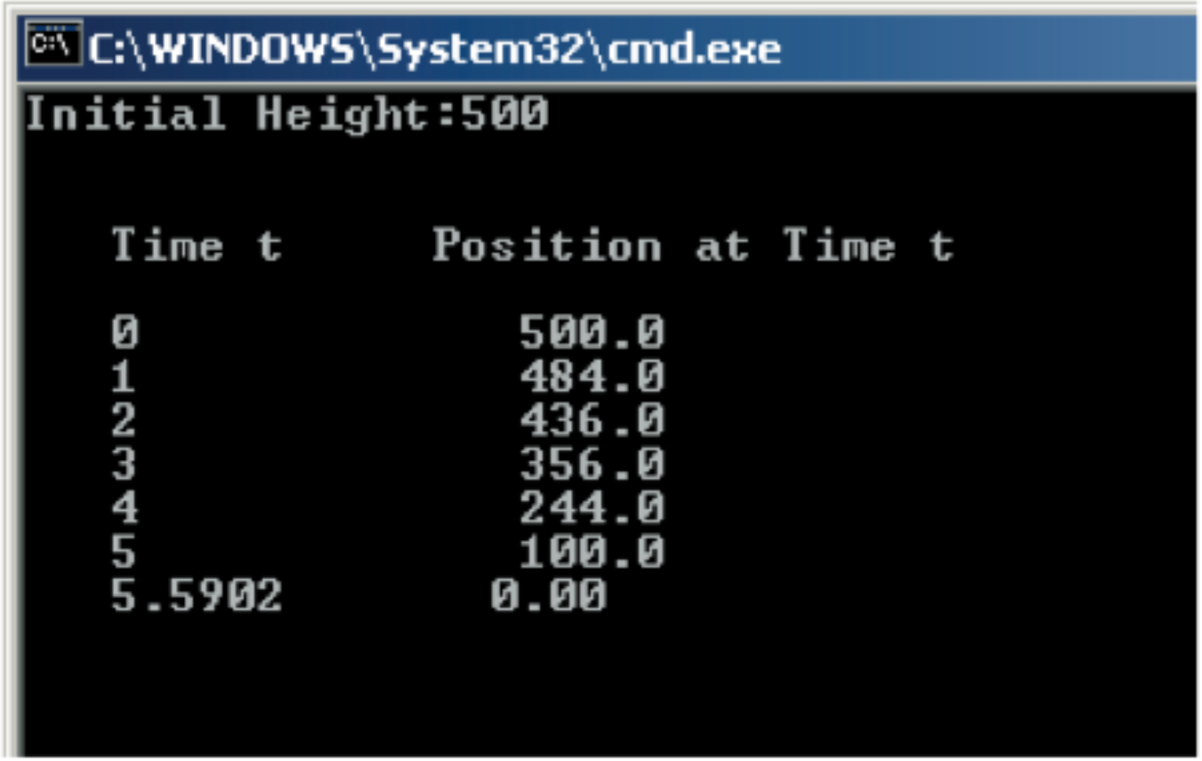
The for Statement



```
      touchTime       = Math.sqrt(initialHeight / 16.0);
      touchTime       = Math.round(touchTime * 10000.0) / 10000.0;
                          //convert to four decimal places

      System.out.println("\n\n   Time t     Position at Time t \n");

      for (int time = 0; time < touchTime; time++) {
          position = -16.0 * time*time + initialHeight;
          System.out.print("    " + time);
          System.out.println("              " + position);
      }

      //print the last second
      System.out.println("    " + touchTime + "      0.00");
   }
}
```

The for Statement

– Output



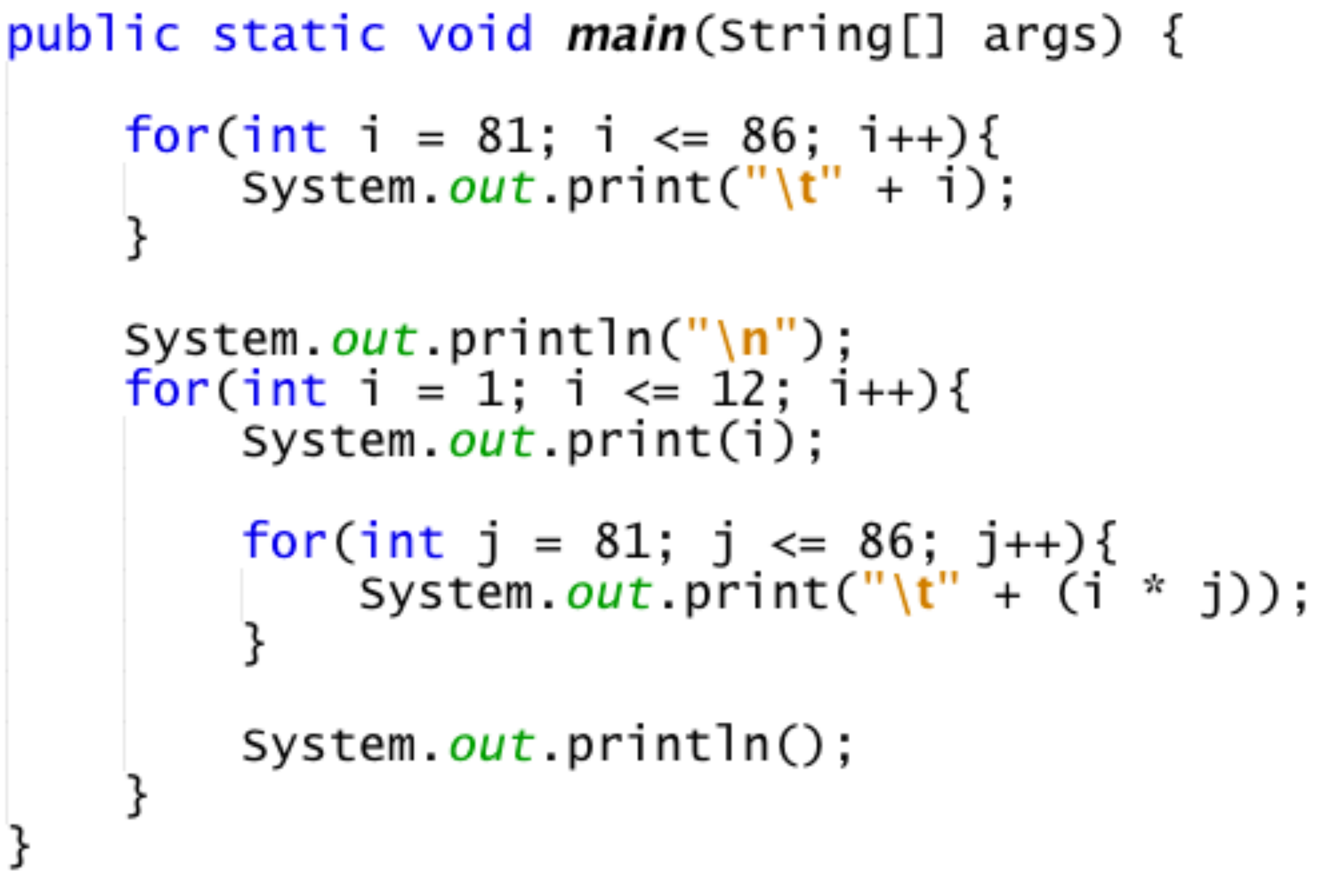
```
Initial Height:500


   Time t        Position at Time t

   0                500.0
   1                484.0
   2                436.0
   3                356.0
   4                244.0
   5                100.0
   5.5902          0.00
```

Nested for Statements

- In many processing tasks, we need to place a for statement inside another for statement. In this section, we introduce a simple nested for statement.

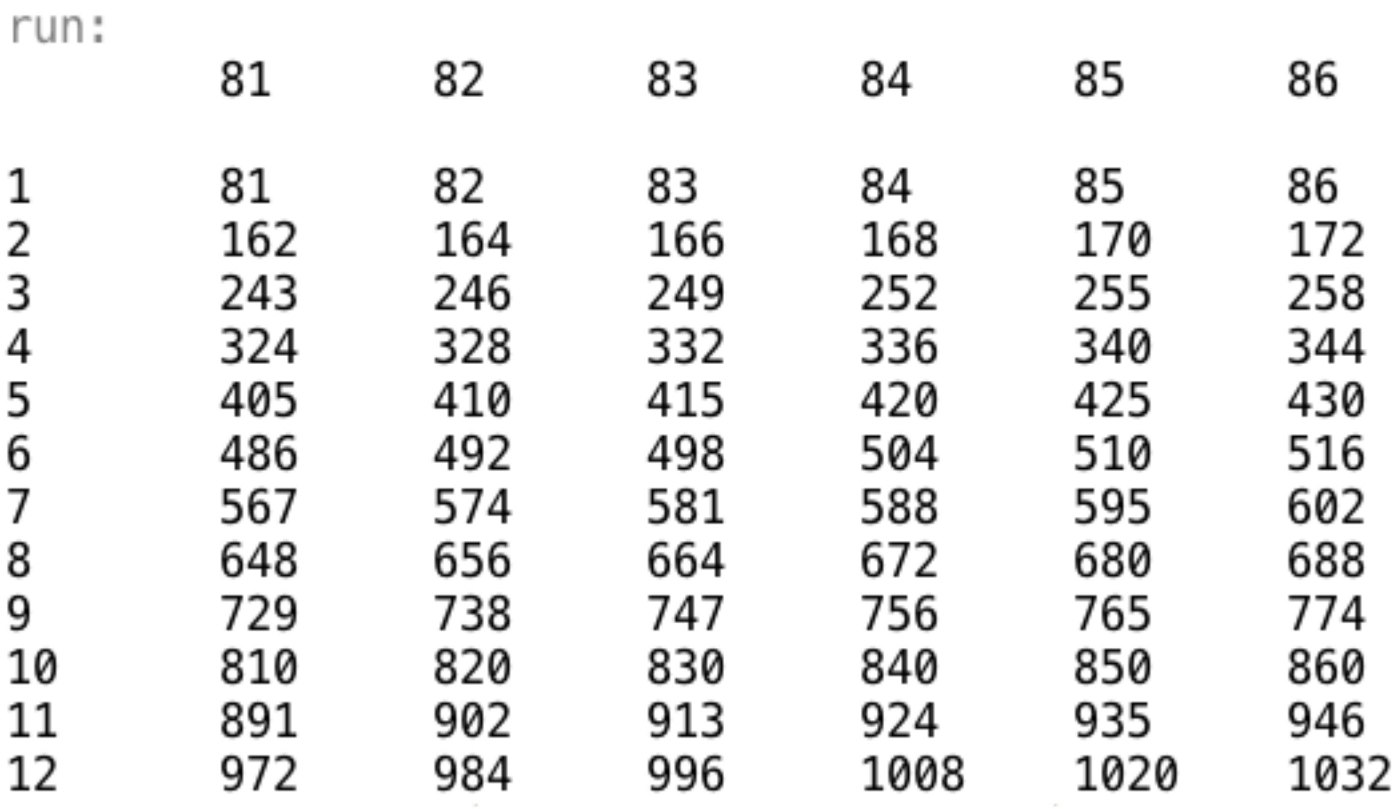
Nested for Statements

- Ex. Create multiplication table



```java
public static void main(String[] args) {

    for(int i = 81; i <= 86; i++){
        System.out.print("\t" + i);
    }

    System.out.println("\n");
    for(int i = 1; i <= 12; i++){
        System.out.print(i);

        for(int j = 81; j <= 86; j++){
            System.out.print("\t" + (i * j));
        }

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

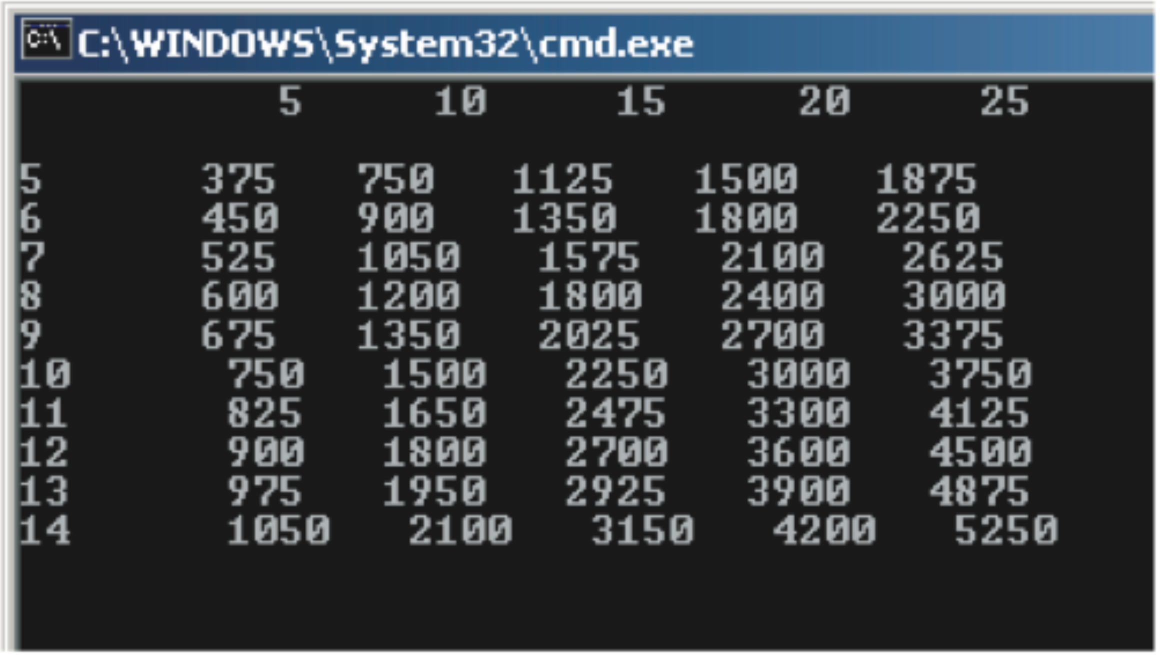
Nested for Statements



```
run:
        81      82      83      84      85      86

1       81      82      83      84      85      86
2       162     164     166     168     170     172
3       243     246     249     252     255     258
4       324     328     332     336     340     344
5       405     410     415     420     425     430
6       486     492     498     504     510     516
7       567     574     581     588     595     602
8       648     656     664     672     680     688
9       729     738     747     756     765     774
10      810     820     830     840     850     860
11      891     902     913     924     935     946
12      972     984     996     1008    1020    1032
```

Formatting Output

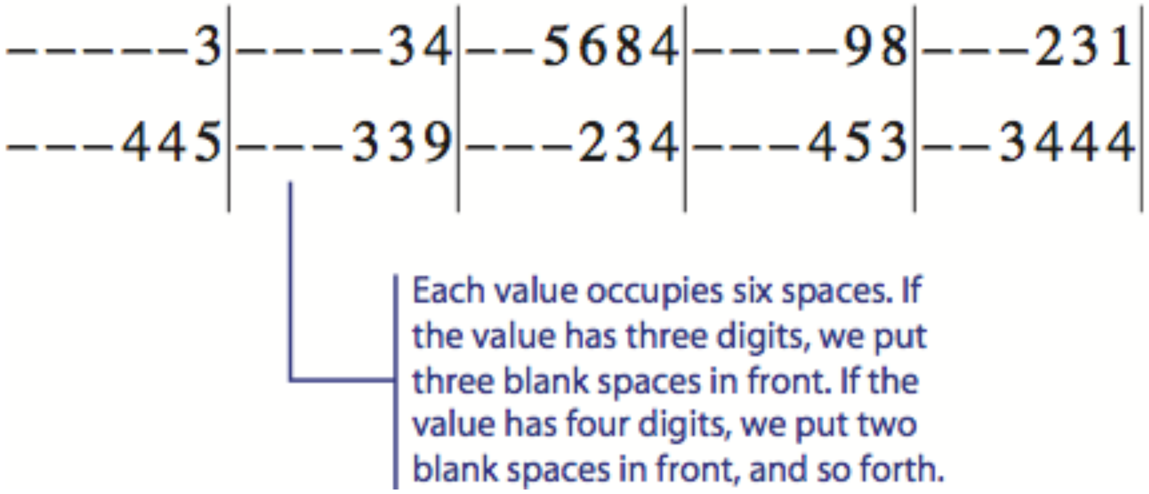
- Sometime we need to format the output so the values are printed out with the proper alignment.



```
          5     10     15     20     25

5      375   750   1125   1500   1875
6      450   900   1350   1800   2250
7      525   1050   1575   2100   2625
8      600   1200   1800   2400   3000
9      675   1350   2025   2700   3375
10      750   1500   2250   3000   3750
11      825   1650   2475   3300   4125
12      900   1800   2700   3600   4500
13      975   1950   2925   3900   4875
14      1050   2100   3150   4200   5250
```

Formatting Output

- The basic idea of formatted output is to allocate the same amount of space for the output values and align the values within the allocated space.



```
-----3|----34|--5684|----98|---231|
---445|---339|---234|---453|--3444|
```

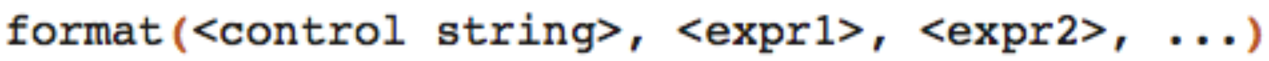
Each value occupies six spaces. If the value has three digits, we put three blank spaces in front. If the value has four digits, we put two blank spaces in front, and so forth.

Formatting Output

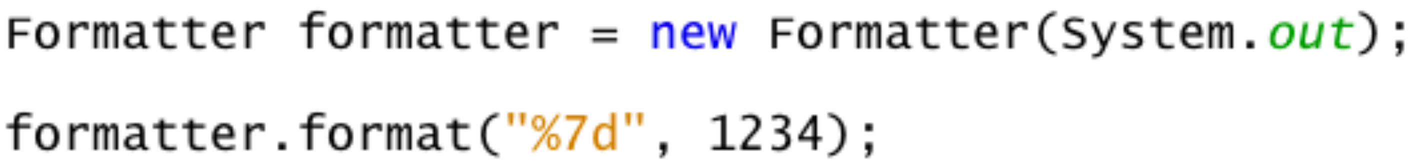
- 2 ways to formatting output
  1. Using Formatter object's format() method
  2. Using System.out.format()

Formatting Output

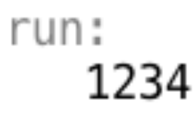
- Using Formatter object's format() method



```
format(<control string>, <expr1>, <expr2>, ...)
```



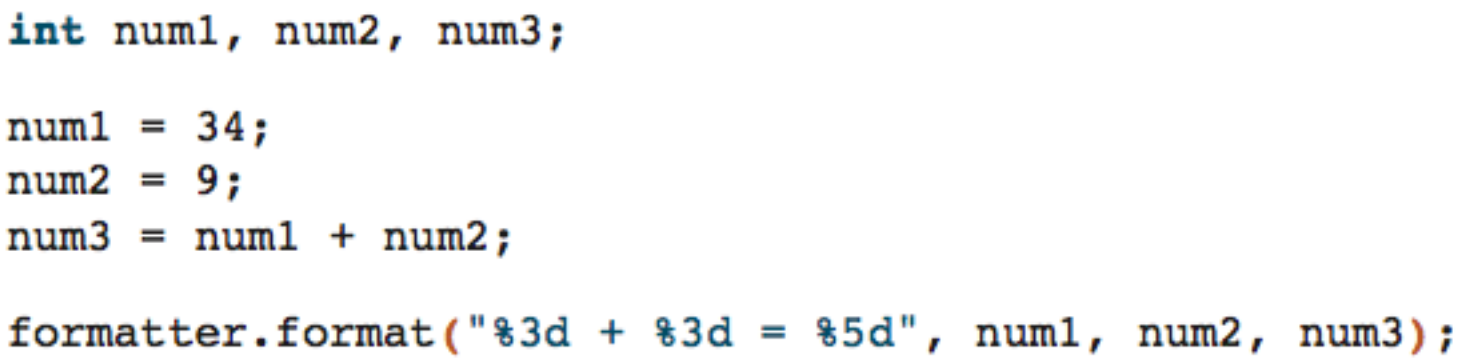
```
Formatter formatter = new Formatter(System.out);

formatter.format("%7d", 1234);
```



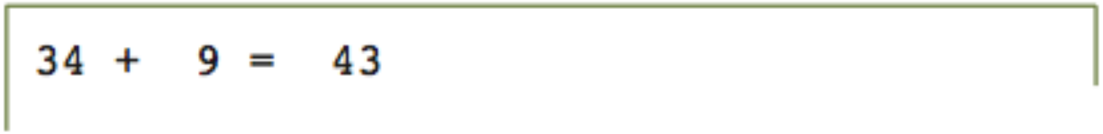
```
run:
   1234
```

Formatting Output

- Ex. Using Formatter object 1

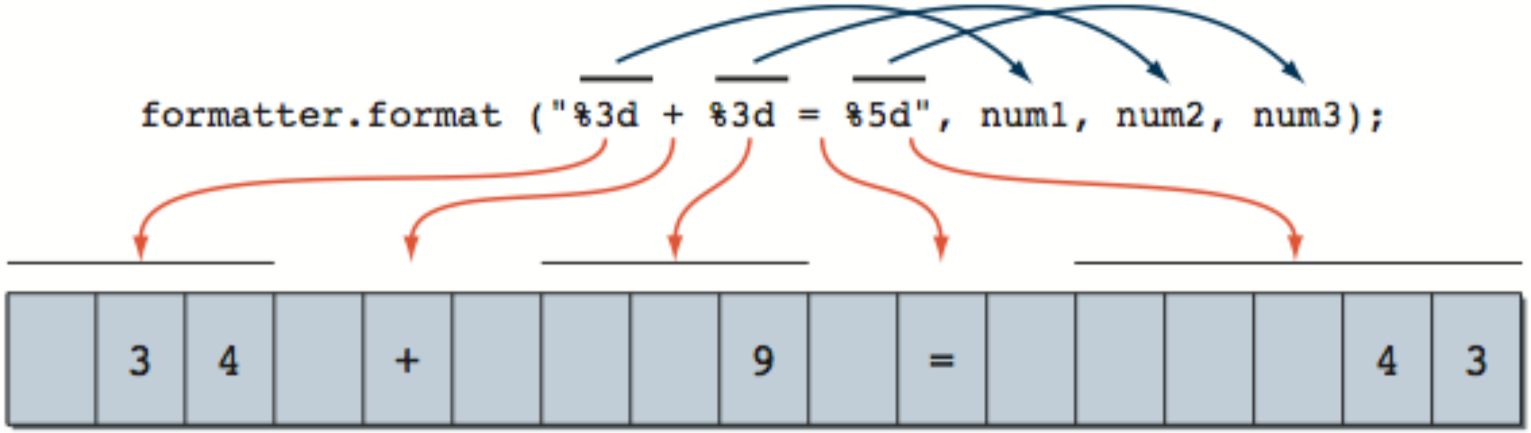


```
int num1, num2, num3;

num1 = 34;
num2 = 9;
num3 = num1 + num2;

formatter.format("%3d + %3d = %5d", num1, num2, num3);
```



```
34 +   9 =    43
```

Formatting Output

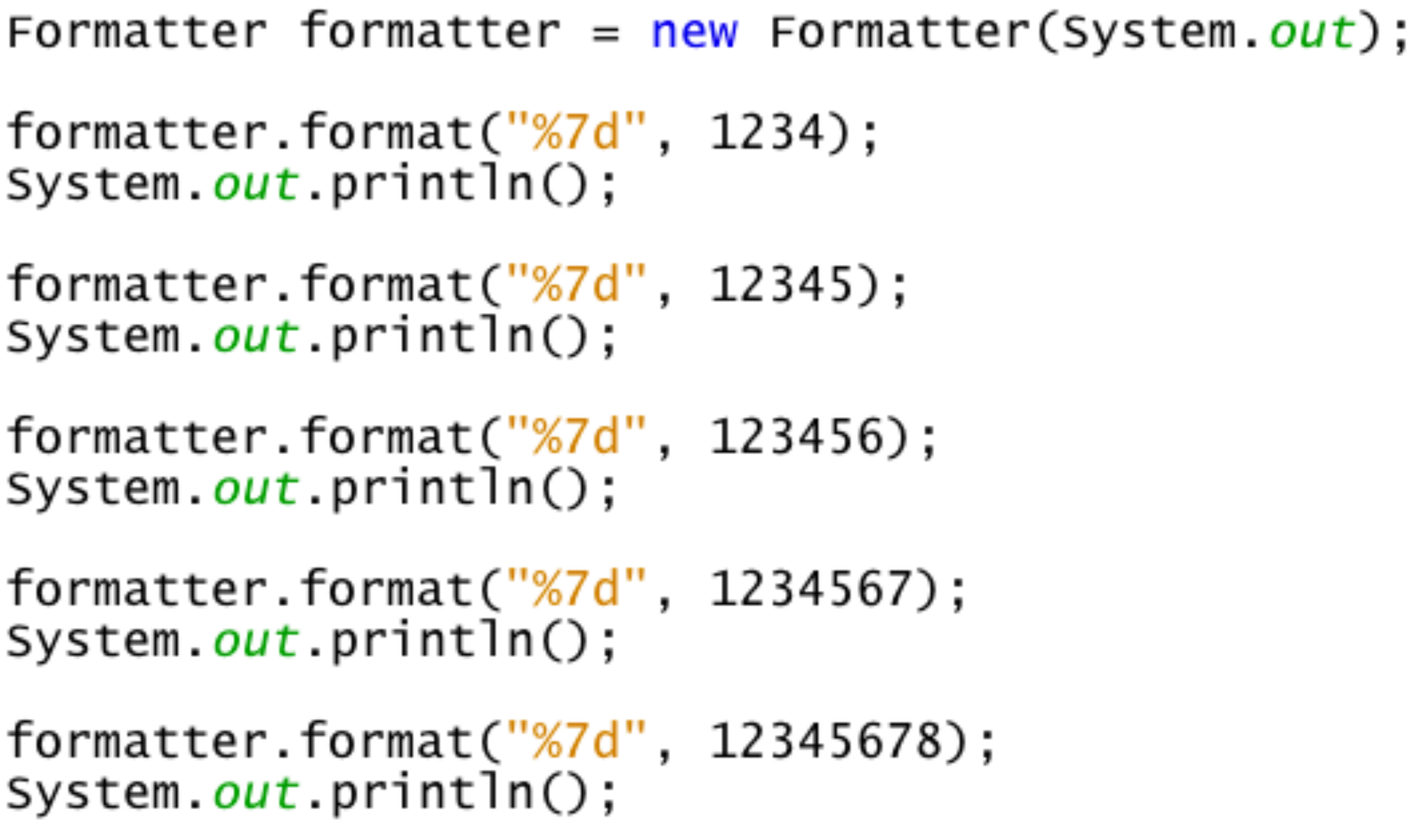


```
formatter.format ("%3d + %3d = %5d", num1, num2, num3);
```

```
 34 +   9 =    43
```

Formatting Output

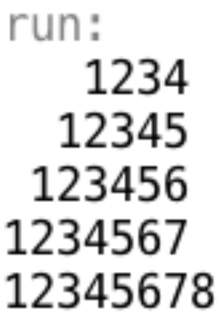
- Ex. Using Formatter object 2



```
Formatter formatter = new Formatter(System.out);

formatter.format("%7d", 1234);
System.out.println();

formatter.format("%7d", 12345);
System.out.println();

formatter.format("%7d", 123456);
System.out.println();

formatter.format("%7d", 1234567);
System.out.println();

formatter.format("%7d", 12345678);
System.out.println();
```

Formatting Output

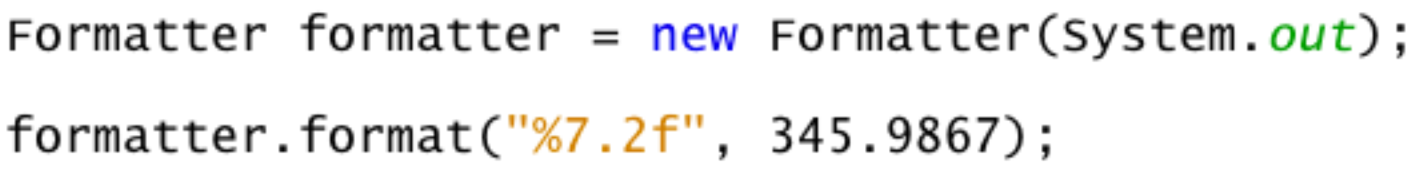
– Output:



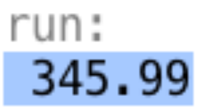
```
run:
   1234
  12345
 123456
1234567
12345678
```

Formatting Output

- We can use Formatter object to format floating-point, String and Date
  - Floating-point



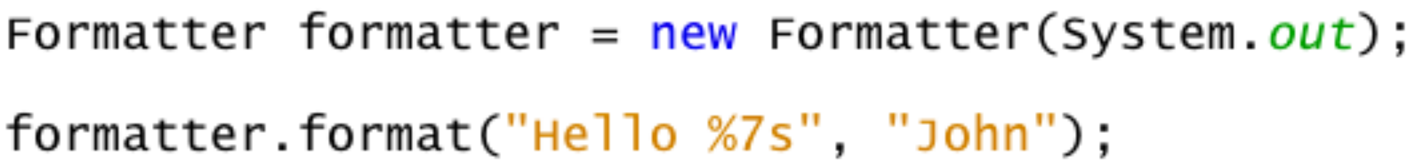
```
Formatter formatter = new Formatter(System.out);

formatter.format("%7.2f", 345.9867);
```



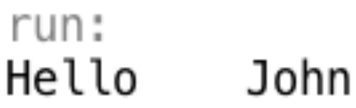
```
run:
 345.99
```

Formatting Output

– String



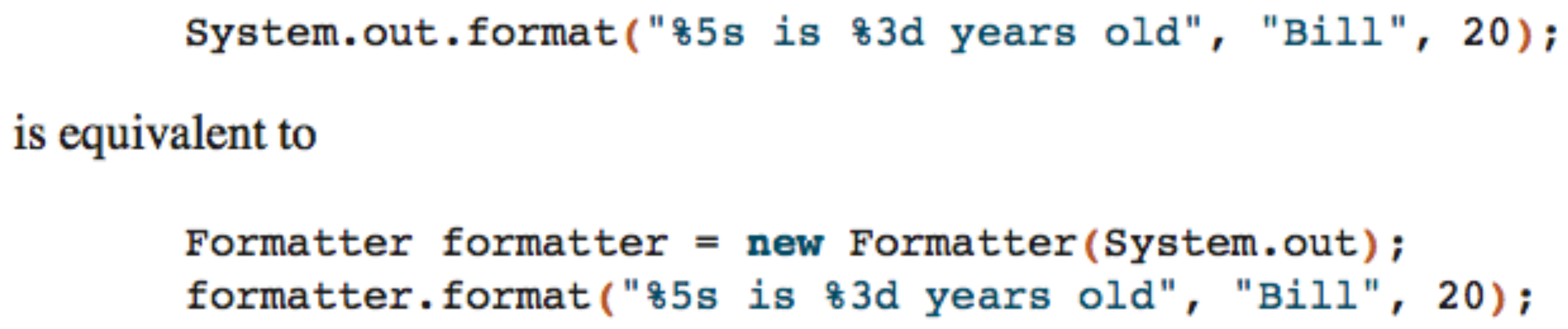
```java
Formatter formatter = new Formatter(System.out);

formatter.format("Hello %7s", "John");
```



```
run:
Hello    John
```

Formatting Output

- Using System.out.format()
  - We can use System.out.format() like using Formatter object's format() method



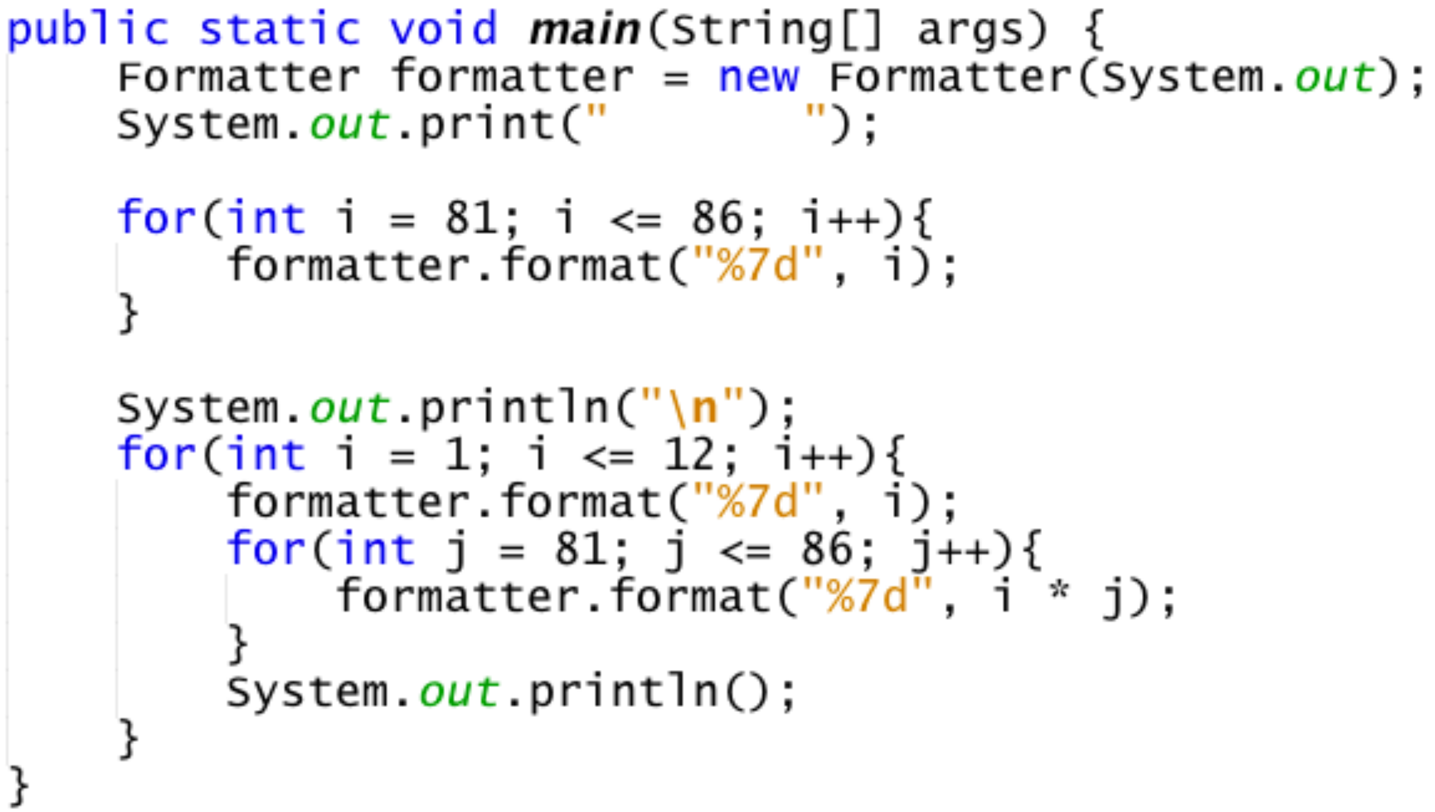
```
System.out.format("%5s is %3d years old", "Bill", 20);
```

is equivalent to

```
Formatter formatter = new Formatter(System.out);
formatter.format("%5s is %3d years old", "Bill", 20);
```

Formatting Output

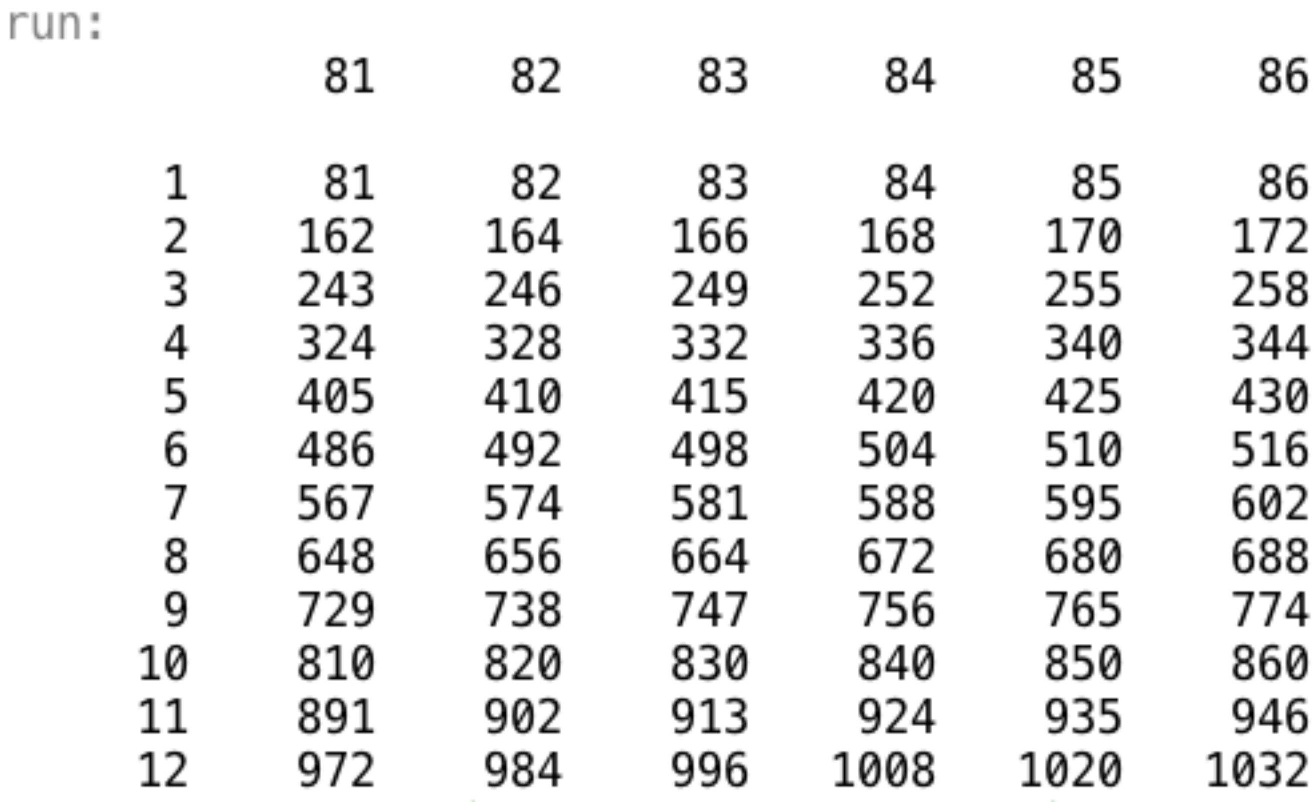
- Ex. Using Formatter object to format multiplication table.



```java
public static void main(String[] args) {
    Formatter formatter = new Formatter(System.out);
    System.out.print("       ");

    for(int i = 81; i <= 86; i++){
        formatter.format("%7d", i);
    }

    System.out.println("\n");
    for(int i = 1; i <= 12; i++){
        formatter.format("%7d", i);
        for(int j = 81; j <= 86; j++){
            formatter.format("%7d", i * j);
        }
        System.out.println();
    }
}
```

Formatting Output

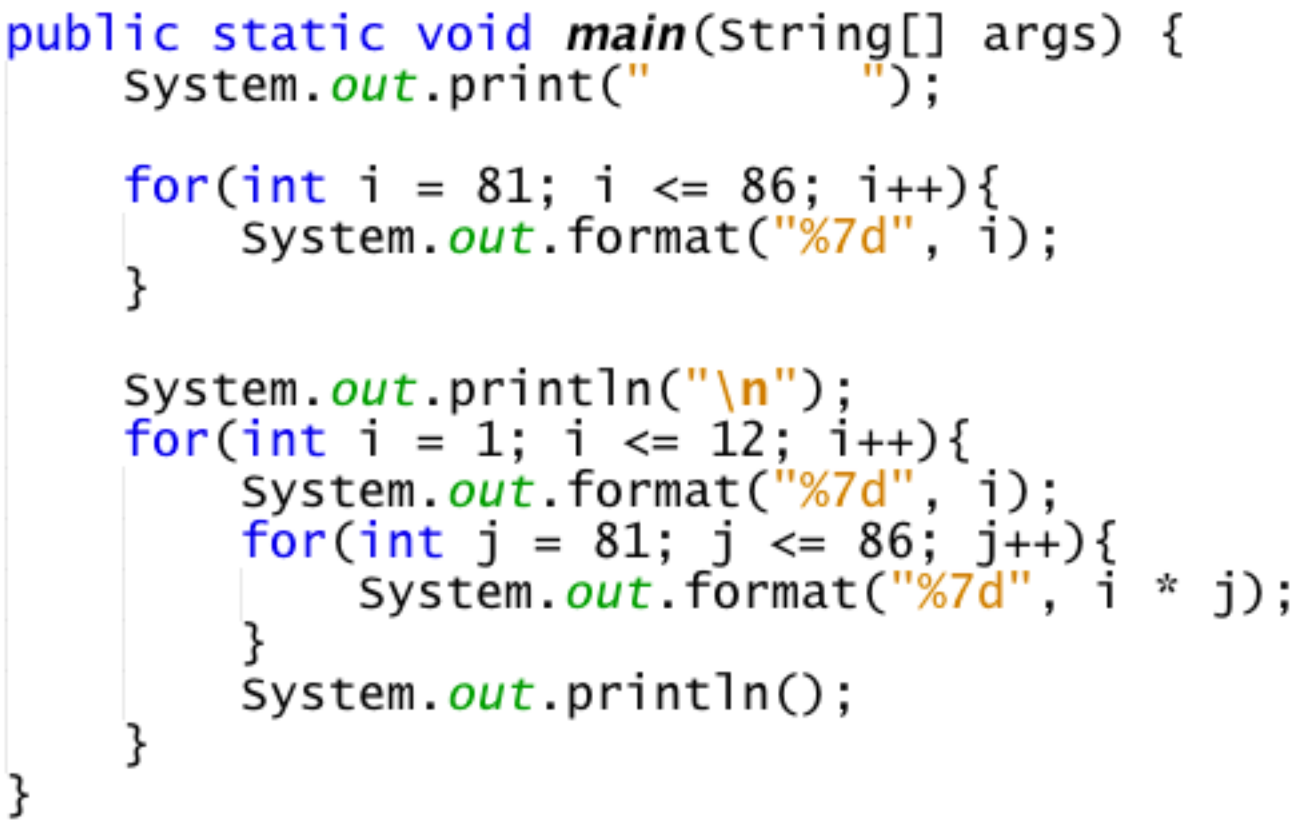
– Output



```
run:
             81     82     83     84     85     86

      1      81     82     83     84     85     86
      2     162    164    166    168    170    172
      3     243    246    249    252    255    258
      4     324    328    332    336    340    344
      5     405    410    415    420    425    430
      6     486    492    498    504    510    516
      7     567    574    581    588    595    602
      8     648    656    664    672    680    688
      9     729    738    747    756    765    774
     10     810    820    830    840    850    860
     11     891    902    913    924    935    946
     12     972    984    996   1008   1020   1032
```

Formatting Output

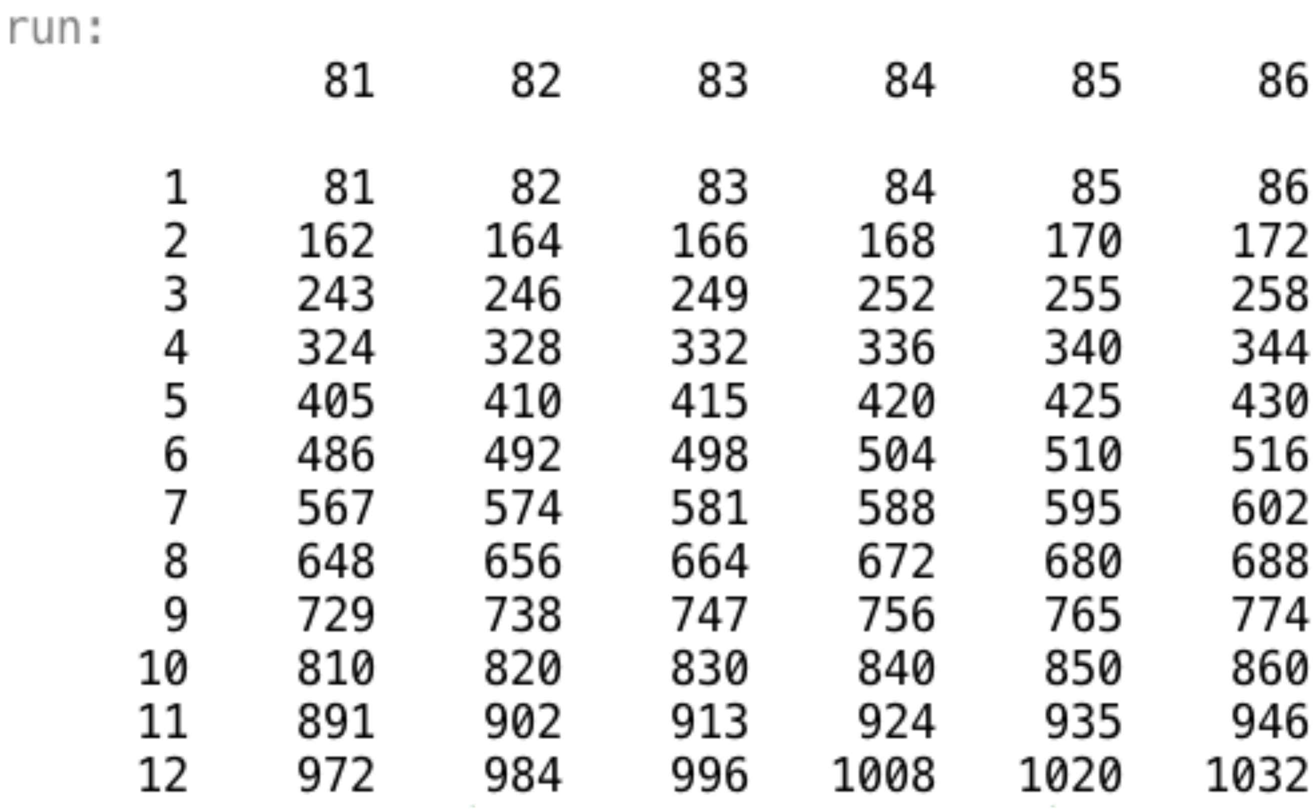
- Ex. Using System.out.format() to format multiplication table.



```java
public static void main(String[] args) {
    System.out.print("       ");

    for(int i = 81; i <= 86; i++){
        System.out.format("%7d", i);
    }

    System.out.println("\n");
    for(int i = 1; i <= 12; i++){
        System.out.format("%7d", i);
        for(int j = 81; j <= 86; j++){
            System.out.format("%7d", i * j);
        }
        System.out.println();
    }
}
```

Formatting Output

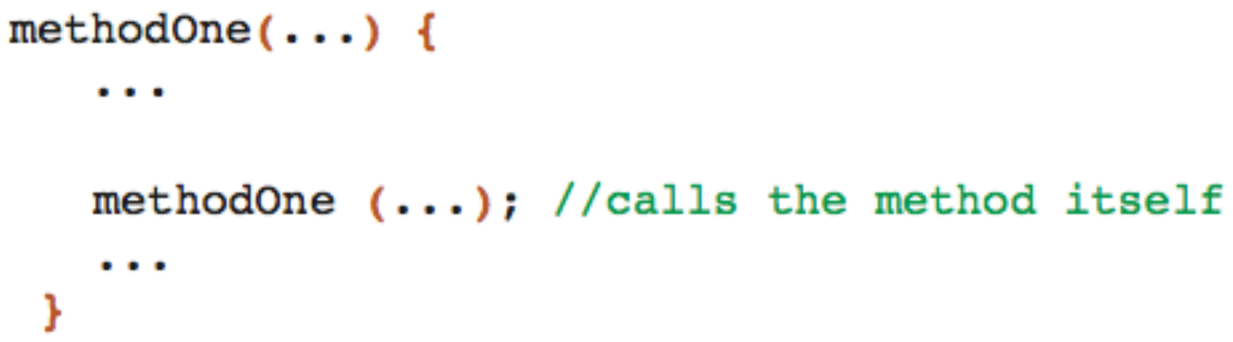
- Output



```
run:
            81     82     83     84     85     86

      1     81     82     83     84     85     86
      2    162    164    166    168    170    172
      3    243    246    249    252    255    258
      4    324    328    332    336    340    344
      5    405    410    415    420    425    430
      6    486    492    498    504    510    516
      7    567    574    581    588    595    602
      8    648    656    664    672    680    688
      9    729    738    747    756    765    774
     10    810    820    830    840    850    860
     11    891    902    913    924    935    946
     12    972    984    996   1008   1020   1032
```

Recursive Methods

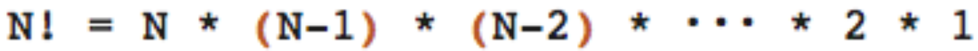
- A recursive method is a method that contains a statement (or statements) that makes a call to itself.



```
methodOne(...) {
    ...

    methodOne (...); //calls the method itself
    ...
  }
```

Recursive Methods

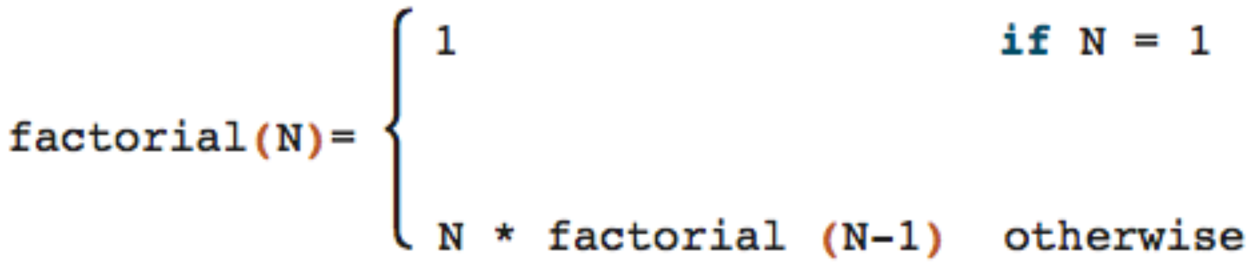
- Ex. Suppose we want to compute the factorial of N. The factorial of N is the product of the first N positive integers



```
N! = N * (N-1) * (N-2) * ··· * 2 * 1
```

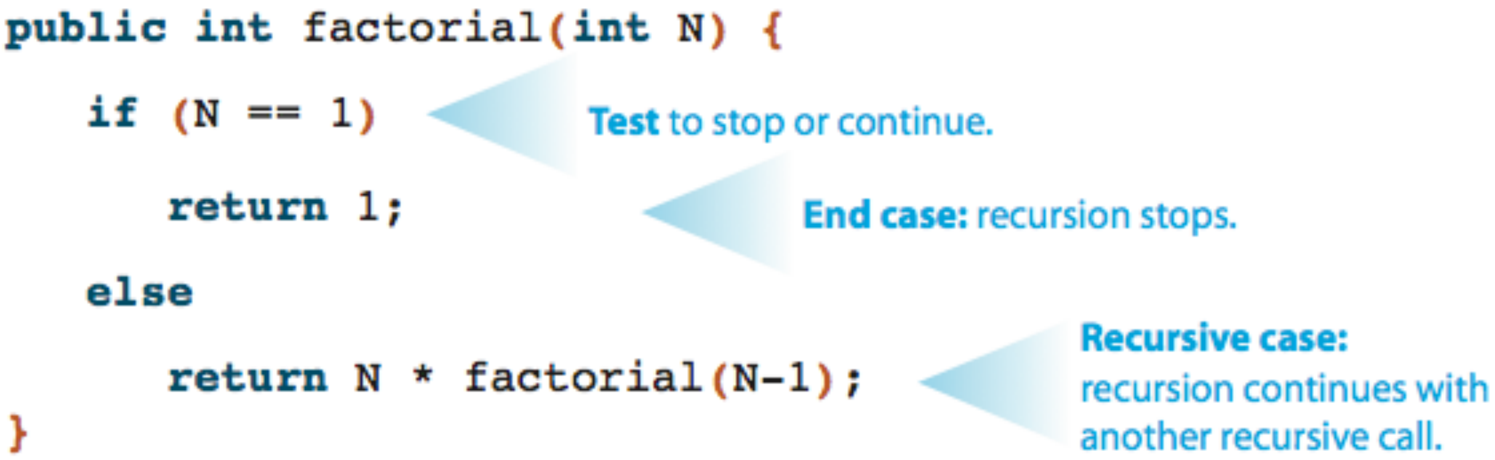
Recursive Methods

– We will write a recursive method to compute the factorial of N. We can define the factorial of N recursively as



```
               ⎧ 1                        if N = 1
factorial(N)=  ⎨
               ⎩ N * factorial (N-1)   otherwise
```

Recursive Methods



```
public int factorial(int N) {
   if (N == 1)
      return 1;
   else
      return N * factorial(N-1);
}
```

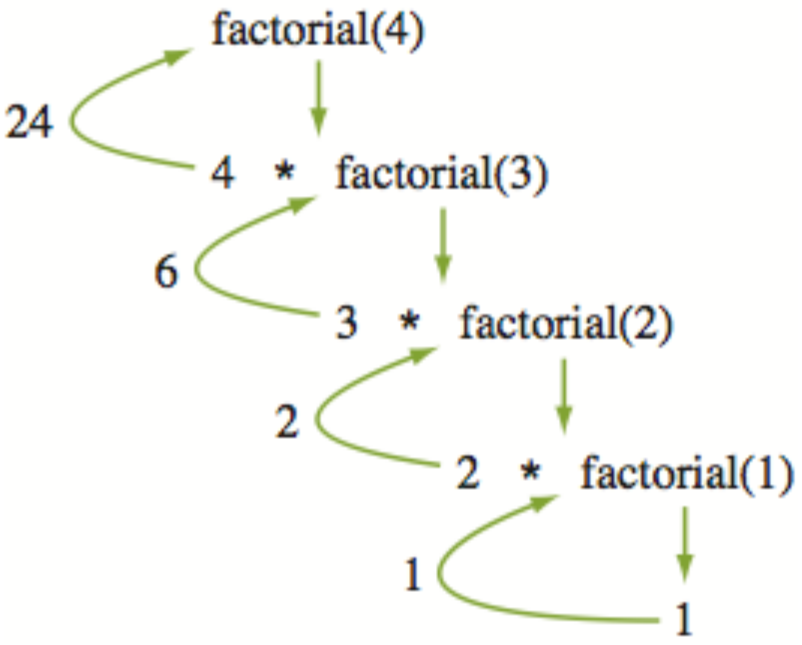
Test to stop or continue.

End case: recursion stops.

Recursive case: recursion continues with another recursive call.

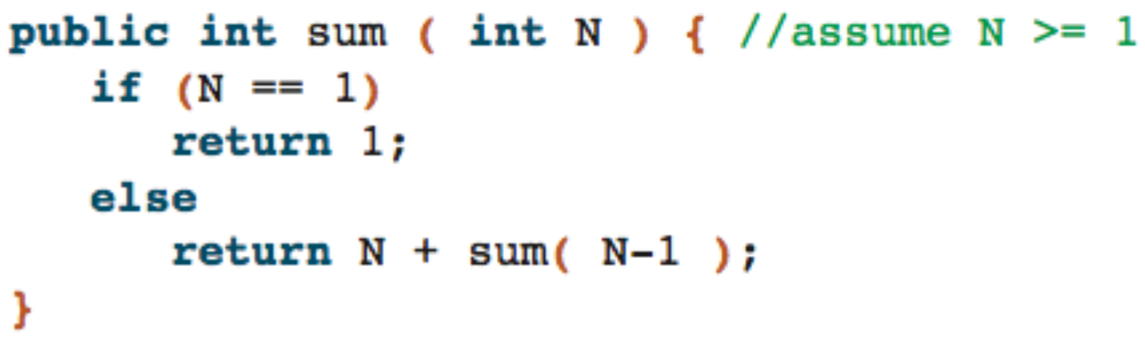
Recursive Methods

– factorial(4) is evaluated as follows:



Recursive Methods

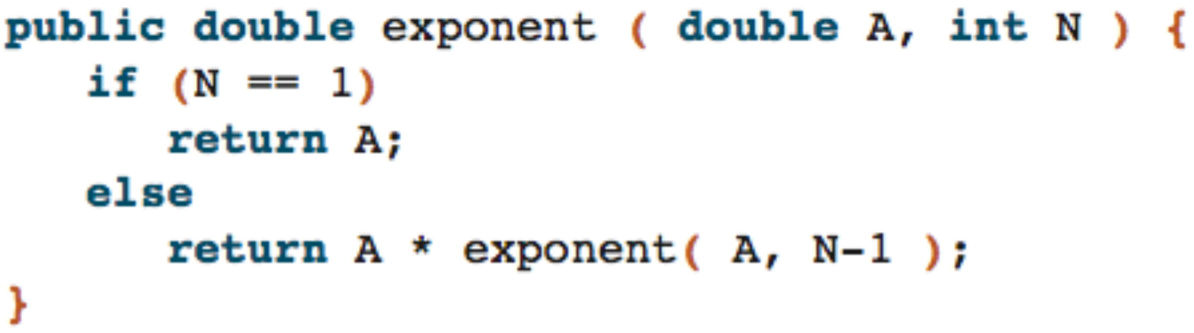
- Ex. Implement sum() method using recursion to computes the sum of the first N positive integers 1, 2, ..., N.



```
public int sum ( int N ) { //assume N >= 1
    if (N == 1)
        return 1;
    else
        return N + sum( N-1 );
}
```

Recursive Methods

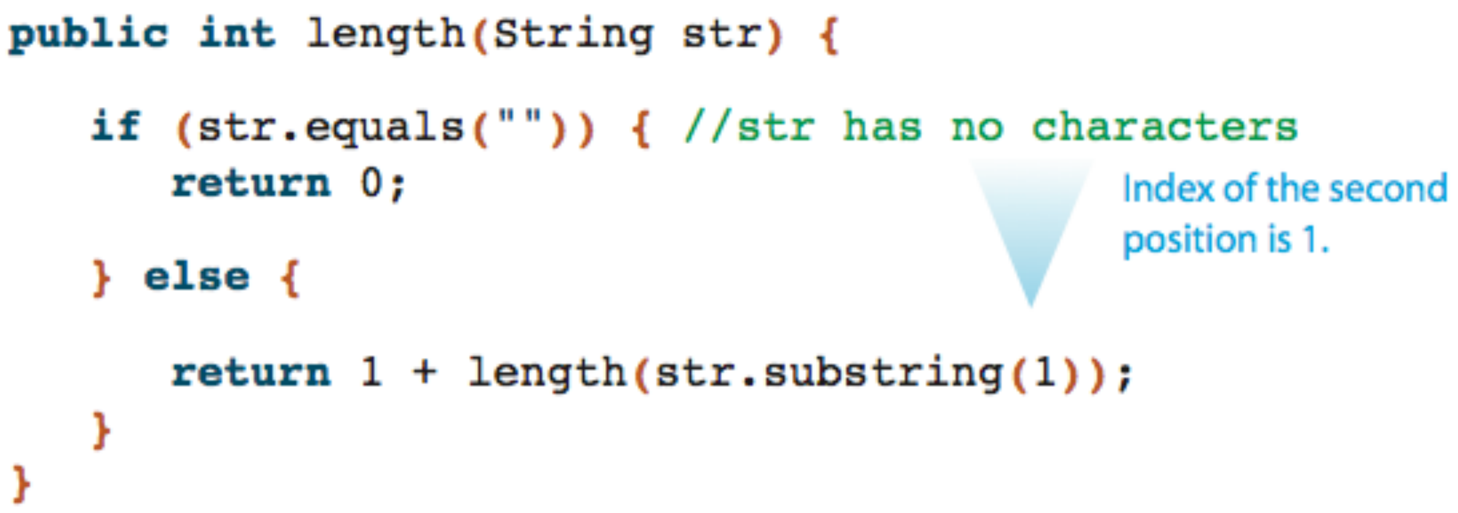
- Ex. Implement exponent() method using recursion to computes the exponentiation A^N



```
public double exponent ( double A, int N ) {
    if (N == 1)
        return A;
    else
        return A * exponent( A, N-1 );
}
```

Recursive Methods

- Ex. Implement String’s length() method using recursion to computes length of String



```
public int length(String str) {

    if (str.equals("")) { //str has no characters
        return 0;

    } else {

        return 1 + length(str.substring(1));
    }
}
```

Index of the second position is 1.

Recursive Methods

- We used factorial, sum, exponentiation, and length as examples to introduce some of the basic concepts of recursion, but we should never actually write these methods using recursion. The methods can be written more efficiently in an iterative manner using a simple for loop.

Summary

- A repetition control statement is used to repeatedly execute a block of code until a certain condition is met.
- Three repetition control statements are while, do–while, and for.
- The while statement is called a ‘pretest loop’, and the do–while statement is called a ‘posttest loop’. The for statement is also a ‘pretest loop’.

Summary

- The loop-and-a-half repetition control is the most general way of writing a loop. The break statement is used within the loop body to exit the loop when a certain condition is met.
- The nested for statement is used very often because it is ideally suited to process tabular data.
- Output values can be formatted by using the Formatter class or System.out.format().

Reference

- C. Thomas Wu, An Introduction to Object-Oriented Programming with Java, 5th Edition
  - Chapter 6: Repetition Statements

Question?