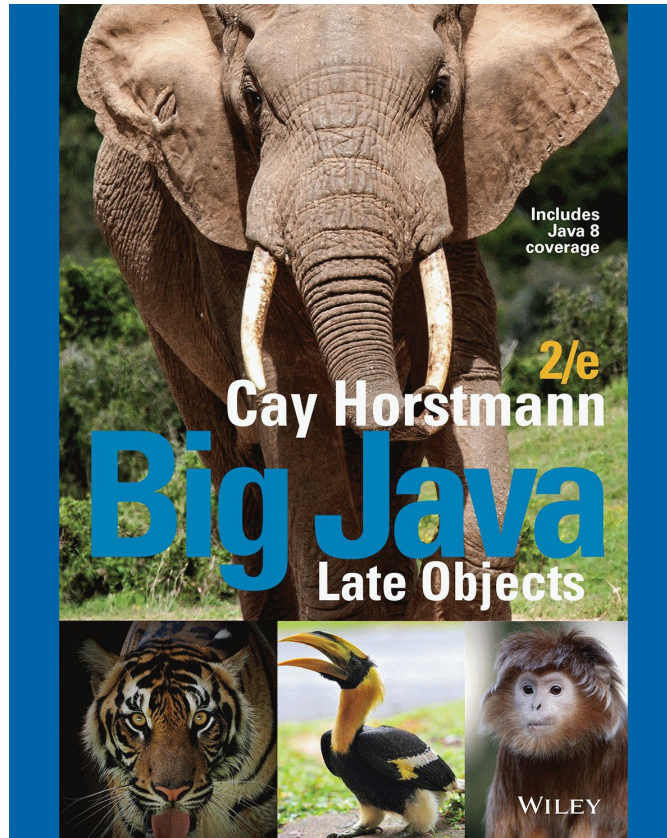


# Chapter 4 - Loops

---



# 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

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

Add 1 to the year value.

Compute the interest as  $\text{balance} \times 0.05$  (i.e, 5 percent interest).

Add the interest to the balance.

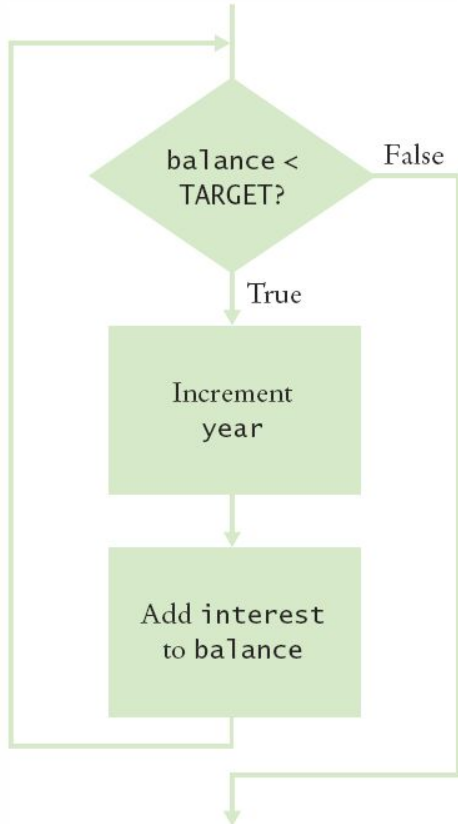
Report the final year value as the answer.

Steps



# 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;
}
```

## Syntax 4.1 **while** Statement

**Syntax**    **while** (*condition*)  
          {  
              *statements*  
          }

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

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



See Common Error 4.2.

double balance = 0;

.

.

.

**while** (balance < TARGET)

{

double interest = balance \* RATE / 100;

balance = balance + interest;

}

This variable is created in each loop iteration.

Beware of "off-by-one" errors in the loop condition.  
 See Common Error 4.3.

Don't put a semicolon here!  
 See Common Error 3.1.

These statements are executed while the condition is true.

Lining up braces is a good idea.  
See Programming Tip 3.1.

Braces are not required if the body contains a single statement, but it's good to always use them.  
 See Programming Tip 3.2.

# Execution of the Loop

1 Check the loop condition

balance = 10000

```
while (balance < TARGET)
{
```

The condition is true

4 After 15 iterations

balance = 20789.28

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

The condition is no longer true

2

year = 15

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);
```

3

balance = 10500

year = 1

```
year++;
double interest = balance * RATE / 100;
balance = balance + interest;
}
```

# DoubleInvestment.java

```
1  /**
2   * This program computes the time required to double an investment.
3   */
4  public class DoubleInvestment
5  {
6      public static void main(String[] args)
7      {
8          final double RATE = 5;
9          final double INITIAL_BALANCE = 10000;
10         final double TARGET = 2 * INITIAL_BALANCE;
11
12         double balance = INITIAL_BALANCE;
13         int year = 0;
14
15         // Count the years required for the investment to double
16
17         while (balance < TARGET)
18         {
19             year++;
20             double interest = balance * RATE / 100;
21             balance = balance + interest;
22         }
23
24         System.out.println("The investment doubled after "
25             + year + " years.");
26     }
27 }
```

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
<pre>i = 0; sum = 0; while (sum &lt; 10) {     i++; sum = sum + i;     <b>Print i and sum;</b> }</pre>	<pre>1 1 2 3 3 6 4 10</pre>	When sum is 10, the loop condition is false, and the loop ends.
<pre>i = 0; sum = 0; while (sum &lt; 10) {     i++; sum = sum - i;     <b>Print i and sum;</b> }</pre>	<pre>1 -1 2 -3 3 -6 4 -10 . . .</pre>	Because sum never reaches 10, this is an “infinite loop” (see Common Error 4.2 on page 132).
<pre>i = 0; sum = 0; while (sum &lt; 0) {     i++; sum = sum - i;     <b>Print i and sum;</b> }</pre>	(No output)	The statement <code>sum &lt; 0</code> is false when the condition is first checked, and the loop is never executed.



## while Loop Examples (2)

---

Loop	Output	Explanation
<pre>i = 0; sum = 0; while (sum &gt;= 10) {     i++; sum = sum + i;     <b>Print i and sum;</b> }</pre>	(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 132).
<pre>i = 0; sum = 0; while (sum &lt; 10) ; {     i++; sum = sum + i;     <b>Print i and sum;</b> }</pre>	(No output, program does not terminate)	Note the semicolon before the {. This loop has an empty body. It runs forever, checking whether <code>sum &lt; 0</code> and doing nothing in the body.

## Self Check 4.1

---

How many years does it take for the investment to triple? Modify the program and run it.

**Answer:** 23 years.

## Self Check 4.2

---

If the interest rate is 10 percent per year, how many years does it take for the investment to double? Modify the program and run it.

**Answer:** 8 years.

## Self Check 4.3

---

Modify the program so that the balance after each year is printed. How did you do that?

**Answer:** Add a statement

`System.out.println(balance);`  
as the last statement in the `while` loop.

## Self Check 4.4

---

Suppose we change the program so that the condition of the `while` loop is

```
while (balance <= TARGET)
```

What is the effect on the program? Why?

**Answer:** The program prints the same output. This is because the balance after 14 years is slightly below \$20,000, and after 15 years, it is slightly above \$20,000.

## Self Check 4.5

---

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	<del>0</del>	
	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
<del>1729</del>	<del>0</del>	
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
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
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
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
<del>17</del>	<del>11</del>	<del>2</del>
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
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
<del>17</del>	<del>11</del>	<del>2</del>
<del>1</del>	<del>18</del>	<del>7</del>
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);
```

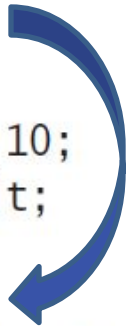
## Tracing Sum of Digits

- 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
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
<del>17</del>	<del>11</del>	<del>2</del>
<del>1</del>	<del>18</del>	<del>7</del>
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 4.6

---

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

--n first decrement then process

n-- first process then decrement

## Self Check 4.7

---

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++;
}
```

first print then `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 4.8

---

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 4.9

---

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
<del>1</del>	1
<del>11</del>	11
<del>21</del>	21
<del>31</del>	31
<del>41</del>	41
<del>51</del>	51
<del>61</del>	61
...	

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

## Self Check 4.10

---

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

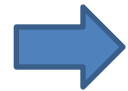
---

- 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



1 Initialize counter

counter = 1

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



2 Check condition

counter = 1

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



3 Execute loop body

counter = 1

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



4 Update counter

counter = 2

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

## Syntax 4.2 **for** Statement

**Syntax**    **for** (*initialization*; *condition*; *update*)  
          {  
              *statements*  
          }

These three  
expressions should be related.  
See Programming Tip 4.1.

This initialization  
happens once  
before the loop starts.

The condition is  
checked before  
each iteration.

This update is  
executed after  
each iteration.

The variable *i*  
is defined only in this  
for loop.

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

This loop executes 6 times.  
See Programming Tip 4.3.

don't have to fill all

# 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:

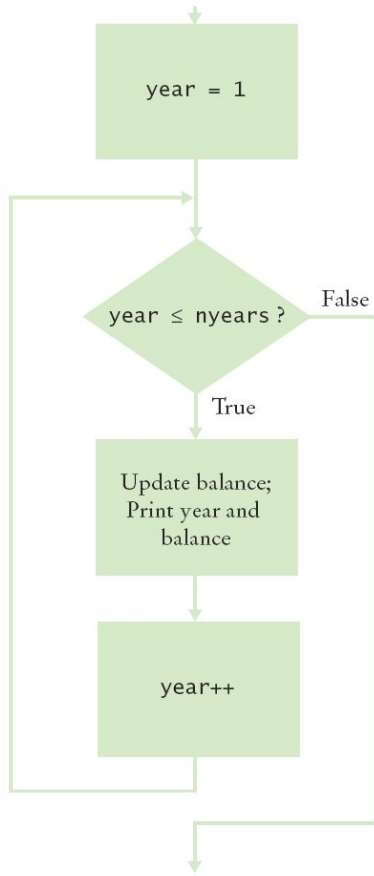
```
for (initialization; condition; update)  
{  
    statements  
}
```

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

```
1  import java.util.Scanner;
2
3  /**
4   * This program prints a table showing the growth of an investment.
5   */
6  public class InvestmentTable
7  {
8      public static void main(String[] args)
9      {
10         final double RATE = 5;
11         final double INITIAL_BALANCE = 10000;
12         double balance = INITIAL_BALANCE;
13
14         System.out.print("Enter number of years: ");
15         Scanner in = new Scanner(System.in);
16         int nyears = in.nextInt();
17
18         // Print the table of balances for each year
19
20         for (int year = 1; year <= nyears; year++)
21         {
22             double interest = balance * RATE / 100;
23             balance = balance + interest;
24             System.out.printf("%4d %10.2f\n", year, balance);
25         }
26     }
27 }
```

# 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. (See Programming Tip 4.4 on page 153.)
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 ith 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 4.11

---

Write the `for` loop of the `InvestmentTable.java` program as a `while` loop.

**Answer:**

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

```
int year = 0
while (year < nyears)
```

## Self Check 4.12

---

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

## Self Check 4.13

---

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);
}
```

## Self Check 4.14

---

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;
}
```

## Self Check 4.15

---

How would you modify the `for` loop of the `InvestmentTable.java` program to print all balances until the investment has doubled?

**Answer:**

```
for (int year = 1;  
     balance <= 2 * INITIAL_BALANCE; year++)
```

However, it is best not to use a `for` loop in this case because the loop condition does not relate to the `year` variable. A `while` loop would be a better choice.

# 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**
- When different conditional operators are used for both bounds, the bounds are called **asymmetric**
- Both kinds of bounds can be appropriate in loops
- Asymmetric bounds work well with strings

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

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

```
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?

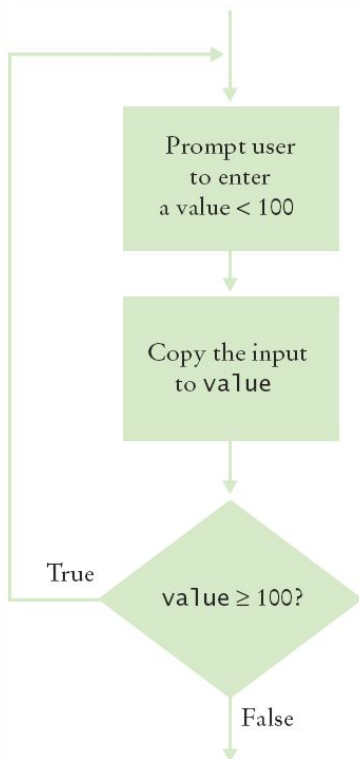
```
final int RAILS = 5;  
for (int i = 1; i < RAILS; i++)  
{  
    System.out.println("Painting rail " + i);  
}
```

```
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 4.16

---

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);
```

## Self Check 4.17

---

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 4.18

---

Suppose Java didn't have a `do` loop. Could you rewrite any `do` loop as a `while` loop?

**Answer:** Yes. The `do` loop

`do { body } while (condition);`

is equivalent to this `while` loop:

```
boolean first = true;
```

```
while (first || condition)
```

```
{
```

```
    body;
```

```
    first = false;
```

```
}
```

## Self Check 4.19

---

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);
```

## Self Check 4.20

---

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);
```

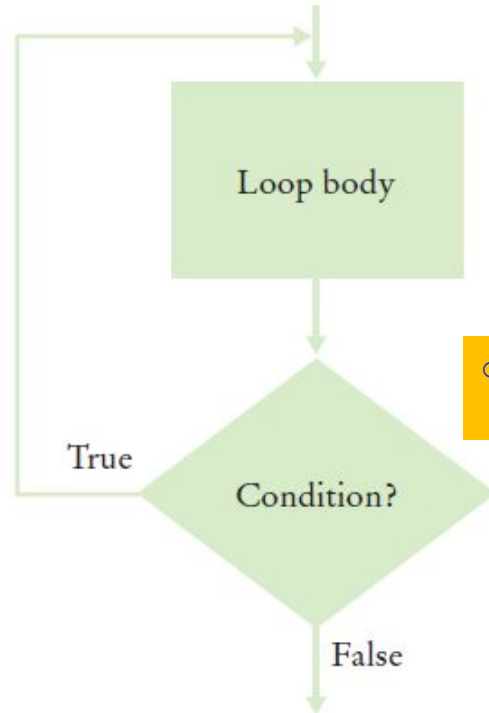
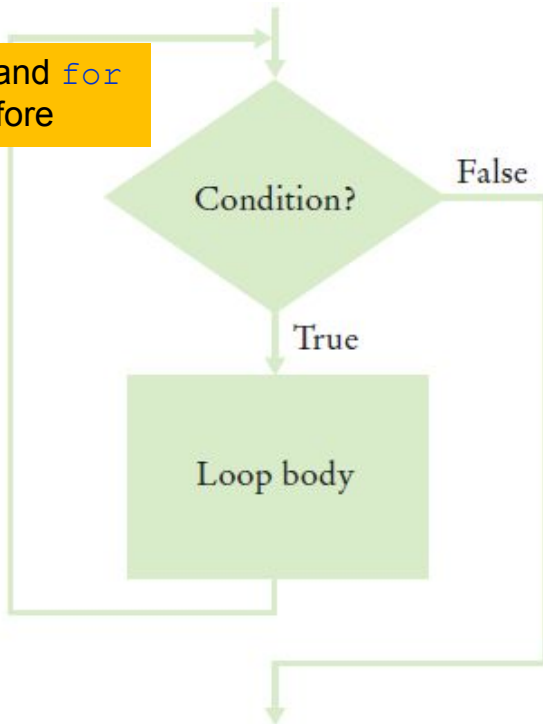


# Programming Tip

---

- 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)

```
8 public static void main(String[] args)
9 {
10     double sum = 0;
11     int count = 0;
12     double salary = 0;
13     System.out.print("Enter salaries, -1 to finish: ");
14     Scanner in = new Scanner(System.in);
15
16     // Process data until the sentinel is entered
17
18     while (salary != -1)
19     {
20         salary = in.nextDouble();
21         if (salary != -1)
22         {
23             sum = sum + salary;
24             count++;
25         }
26     }
27 }
```

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) Prevent divide by 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 }
```

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)                when done is false execute the loop body
{
    value = in.nextDouble();
    if (value == -1)
    {
        done = true;         exit the loop
    }
    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 4.21

---

What does the `SentinelDemo.java` program print when the user immediately types `-1` when prompted for a value?

**Answer:** No Data



## Self Check 4.22

---

Why does the `SentinelDemo.java` program have two checks of the form  
`salary != -1`

**Answer:** The first check ends the loop after the sentinel has been read. The second check ensures that the sentinel is not processed as an input value.

## Self Check 4.23

---

What would happen if the declaration of the `salary` variable in `SentinelDemo.java` was changed to

```
double salary = -1;
```

**Answer:** The `while` loop would never be entered. The user would never be prompted for input. Because `count` stays 0, the program would then print "No data".

## Self Check 4.24

---

In the last example of this section, we prompt the user “Enter values, Q to quit.” What happens when the user enters a different letter?

**Answer:** The `nextDouble` method also returns `false`. A more accurate prompt would have been: “Enter values, a key other than a digit to quit.” But that might be more confusing to the program user who would need to ponder which key to choose.

## Self Check 4.25

---

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.

# Storyboards

---

- One useful problem solving technique is the use of storyboards to model user interaction. It can help answer:
  - What information does the user provide, and in which order?
  - What information will your program display, and in which format?
  - What should happen when there is an error?
  - When does the program quit?

# Storyboard Example

---

- Goal: Converting a sequence of values
  - Will require a loop and some variables
  - Handle one conversion each time through the loop

## Converting a Sequence of Values

What unit do you want to convert from? **cm**

What unit do you want to convert to? **in**

Enter values, terminated by zero ————— Allows conversion of multiple values

**30**

30 cm = 11.81 in

**100**

100 cm = 39.37 in

**0**

What unit do you want to convert from?

Format makes clear what got converted

# What Can Go Wrong?

---

- Unknown unit types
  - How do you spell centimeters and inches?
  - What other conversions are available?
  - Solution:
    - Show a list of the acceptable unit types

From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): **cm**

To unit: **in**

 No need to list the units again

# What Else Can Go Wrong?

- How does the user quit the program?

## Exiting the Program

From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): **cm**

To unit: **in**

Enter values, terminated by zero

**30**

30 cm = 11.81 in

**0**

More conversions (y, n)? **n**

(Program exits)

Sentinel triggers the prompt to exit

- Storyboards help you plan a program
  - Knowing the flow helps you structure your code



## Self Check 4.26

---

Provide a storyboard panel for a program that reads a number of test scores and prints the average score. The program only needs to process one set of scores. Don't worry about error handling.

**Answer:** *Computing the average*

```
Enter scores, Q to quit: 90 80 90 100 80 Q
The average is 88
(Program exits)
```

## Self Check 4.27

Google has a simple interface for converting units. You just type the question, and you get the answer.



Make storyboards for an equivalent interface in a Java program. Show a scenario in which all goes well, and show the handling of two kinds of errors.

**Answer:** *Simple conversion*

Only one value can be converted

Your conversion question: How many in are 30 cm

30 cm = 11.81 in

(Program exits) Run program again for another question

*Unknown unit*

Your conversion question: How many inches are 30 cm?

Unknown unit: inches

Known units are in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, ga

(Program exits)

*Program doesn't understand question syntax*

Your conversion question: What is an ångström?

Please formulate your question as "How many (unit) are (value) (unit)?"

(Program exits)

## Self Check 4.28

---

Consider a modification of the program in Self Check 26. Suppose we want to drop the lowest score before computing the average. Provide a storyboard for the situation in which a user only provides one score.

**Answer:** *One score is not enough*

```
Enter scores, Q to quit: 90 Q
Error: At least two scores are required.
(Program exits)
```

## Self Check 4.29

---

What is the problem with implementing the following storyboard in Java?

### Computing Multiple Averages

Enter scores: 90 80 90 100 80

The average is 88

Enter scores: 100 70 70 100 80

The average is 84

Enter scores: -1

(Program exits)

-1 is used as a sentinel to exit the program

**Answer:** It would not be possible to implement this interface using the Java features we have covered up to this point. There is no way for the program to know when the first set of inputs ends. (When you read numbers with `value = in.nextDouble()`, it is your choice whether to put them on a single line or multiple lines.)

## Self Check 4.30

Produce a storyboard for a program that compares the growth of a \$10,000 investment for a given number of years under two interest rates.

**Answer:** *Comparing two interest rates*

First interest rate in percent: 5

Second interest rate in percent: 10

Years: 5

Year	5%	10%
0	10000.00	10000.00
1	10500.00	11000.00
2	11025.00	12100.00
3	11576.25	13310.00
4	12155.06	14641.00
5	12762.82	16105.10

This row clarifies that 1 means the end of the first year

# 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.toUpperCase(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 && execute loop when found is false
      position < str.length())
{
    ch = str.charAt(position);
    if (Character.isLowerCase(ch))
    {
        found = true;    exit loop
    }
    else { position++; }
}
```



```
boolean found = true;
if(found&&...){
    ...}
else{
    found = false;}
```

# 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 4.31

---

What total is computed when no user input is provided in the algorithm in Section 4.7.1?

**Answer:** The total is zero.

## Self Check 4.32

---

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; }
}
```

## Self Check 4.33

---

What are the values of `position` and `ch` when no match is found in the algorithm in Section 4.7.3?

**Answer:** `position` is `str.length()` and `ch` is set to the last character of the string or, if the string is empty, is unchanged from its initial value, `'?'`. Note that `ch` must be initialized with some value—otherwise the compiler will complain about a possibly uninitialized variable.

## Self Check 4.34

---

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 4.35

---

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--; }
}
```

## Self Check 4.36

---

What happens with the algorithm in Section 4.7.6 when no input is provided at all?  
How can you overcome that problem?

**Answer:** The initial call to `in.nextDouble()` fails, terminating the program. One solution is to do all input in the loop and introduce a Boolean variable that checks whether the loop is entered for the first time.

```
double input = 0;
boolean first = true;
while (in.hasNextDouble())
{
    double previous = input;
    input = in.nextDouble();
    if (first) { first = false; }
    else if (input == previous)
    {
        System.out.println("Duplicate input");
    }
}
```

# 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

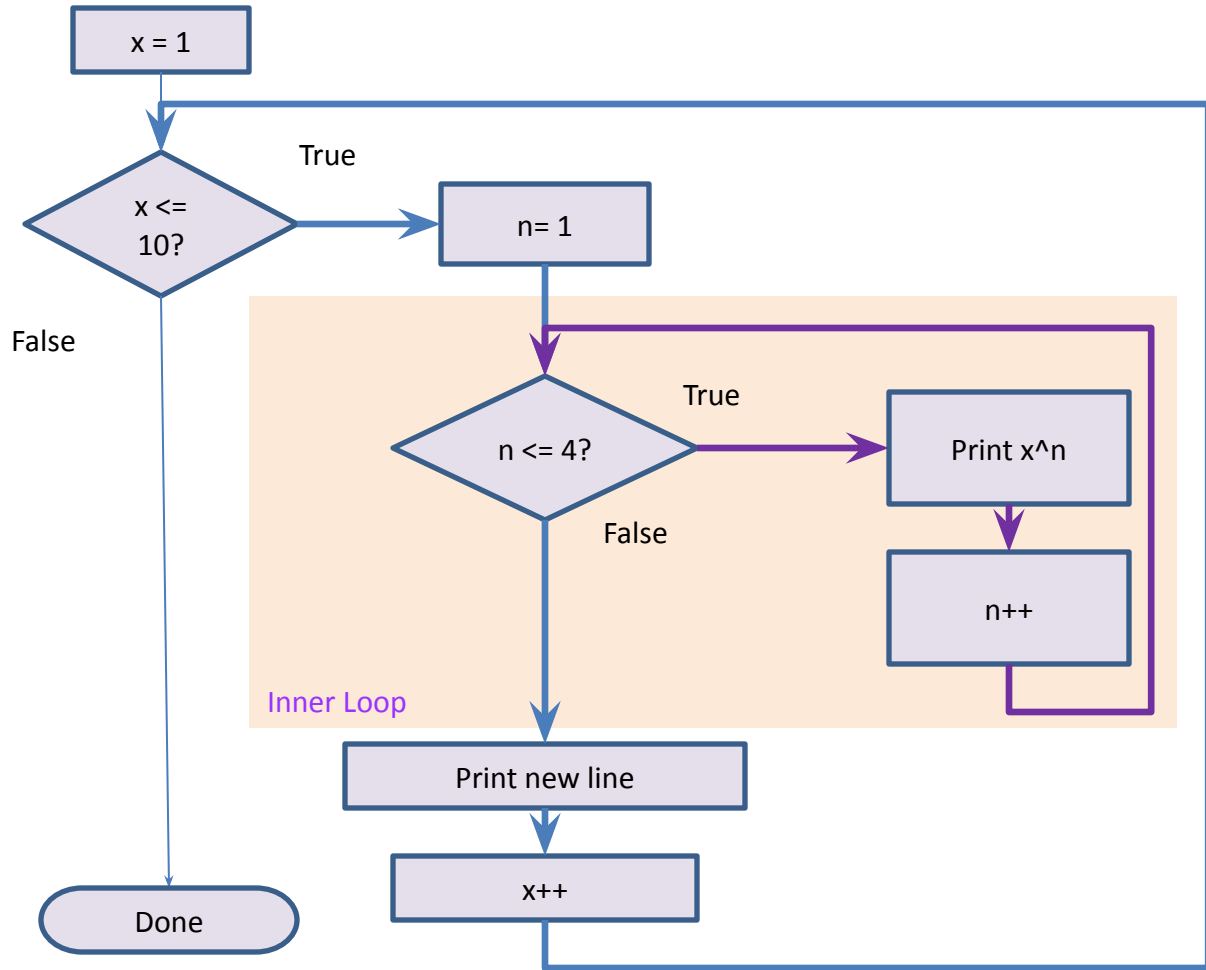
# Nested Loops

---

- 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

```
1  /**
2   * This program prints a table of powers of x.
3   */
4  public class PowerTable
5  {
6      public static void main(String[] args)
7      {
8          final int NMAX = 4;
9          final double XMAX = 10;
10
11         // Print table body
12
13         for (double x = 1; x <= XMAX; x++)
14         {
15             // Print table row
16             Body of outer loop
17
18             for (int n = 1; n <= NMAX; n++)
19             {
20                 System.out.printf("%10.0f", Math.pow(x, n));
21                 Body of inner loop
22
23                 System.out.println();
24                 new line
25             }
26         }
27     }
```

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 4.37

---

Why is there a statement `System.out.println()` ; in the outer loop but not in the inner loop of the `PowerTable.java` program?

**Answer:** All values in the inner loop should be displayed on the same line.

## Self Check 4.38

---

How would you change the program to display all powers from  $x^0$  to  $x^5$ ?

**Answer:** Change lines 13, 18, and 30 to `for (int n = 0; n <= NMAX; n++)`. Change `NMAX` to 5.

## Self Check 4.39

---

If you make the change in Self Check 38, how many values are displayed?

**Answer:** 60: The outer loop is executed 10 times, and the inner loop 6 times.

## Self Check 4.40

---

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
```

## Self Check 4.41

---

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();  
}
```

## Problem Solving: Gallery6.java

```
1 public class Gallery6
2 {
3     public static void main(String[] args)
4     {
5         final int MAX_WIDTH = 720;
6         final int GAP = 10;
7         final int PICTURES = 20;
8
9         Picture pic = new Picture();
10        pic.load("picture1.jpg");
11        int x = 0;
12        int y = 0;
13        int maxY = 0;
14
15        for (int i = 2; i <= 20; i++)
16        {
17            maxY = Math.max(maxY, y + pic.getHeight());
18            Picture previous = pic;
19            pic = new Picture();
20            pic.load("picture" + i + ".jpg");
21            x = x + previous.getWidth() + GAP;
22            if (x + pic.getWidth() >= MAX_WIDTH)
23            {
24                x = 0;
25                y = maxY + GAP;
26            }
27            pic.move(x, y);
28        }
29    }
30 }
```

## Self Check 4.42

---

Suppose you are asked to find all words in which no letter is repeated from a list of words. What simpler problem could you try first?

**Answer:** Of course, there is more than one way to simplify the problem. One way is to print the words in which the first letter is not repeated.

## Self Check 4.43

---

You need to write a program for DNA analysis that checks whether a substring of one string is contained in another string. What simpler problem can you solve first?

**Answer:** You could first write a program that prints all substrings of a given string.



## Self Check 4.44

---

You want to remove “red eyes” from images and are looking for red circles. What simpler problem can you start with?

**Answer:** You can look for a single red pixel, or a block of nine neighboring red pixels.

## Self Check 4.45

---

Consider the task of finding numbers in a string. For example, the string “In 1987, a typical personal computer cost \$3,000 and had 512 kilobytes of RAM.” has three numbers. Break this task down into a sequence of simpler tasks.

**Answer:** Here is one plan:

- a. Find the position of the first digit in a string.
- b. Find the position of the first non-digit after a given position in a string.
- c. Extract the first integer from a string (using the preceding two steps).
- d. Print all integers from a string. (Use the first three steps, then repeat with the substring that starts after the extracted integer.)

# 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

---

```
1  /**
2   * This program prints ten random numbers between 0 and 1.
3   */
4  public class RandomDemo
5  {
6      public static void main(String[] args)
7      {
8          for (int i = 1; i <= 10; i++)
9          {
10             double r = Math.random();
11             System.out.println(r);
12         }
13     }
14 }
```

## 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

```
1  /**
2   * This program simulates tosses of a pair of dice.
3   */
4  public class Dice
5  {
6      public static void main(String[] args)
7      {
8          for (int i = 1; i <= 10; i++)
9          {
10             // Generate two random numbers between 1 and 6
11
12             int d1 = (int) (Math.random() * 6) + 1;
13             int d2 = (int) (Math.random() * 6) + 1;
14             System.out.println(d1 + " " + d2);
15         }
16         System.out.println();
17     }
18 }
```



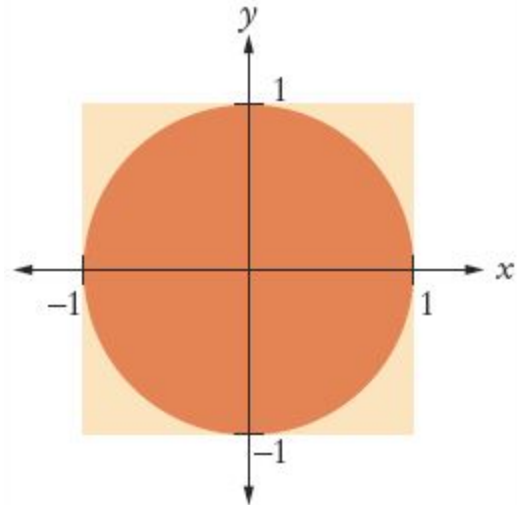
## Program Run

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

# The Monte Carlo Method

---

- Used to find approximate solutions to problems that cannot be precisely solved
- Example: Approximate PI using the relative areas of a circle inside a square
  - Uses simple arithmetic
  - Hits are inside circle
  - Tries are total number of tries
  - Ratio is  $4 \times \text{Hits} / \text{Tries}$



# MonteCarlo.java (1)

```
1  /**
2   * This program computes an estimate of pi by simulating dart throws onto a square.
3   */
4  public class MonteCarlo
5  {
6      public static void main(String[] args)
7      {
8          final int TRIES = 10000;
9
10         int hits = 0;
11         for (int i = 1; i <= TRIES; i++)
12         {
13             // Generate two random numbers between -1 and 1
14
15             double r = Math.random();
16             double x = -1 + 2 * r; // Between -1 and 1
17             r = Math.random();
18             double y = -1 + 2 * r;
19
20             // Check whether the point lies in the unit circle
21
22             if (x * x + y * y <= 1) { hits++; }
23         }
```

## MonteCarlo.java (2)

---

```
24
25      /*
26         The ratio hits / tries is approximately the same as the ratio
27         circle area / square area = pi / 4
28      */
29
30      double piEstimate = 4.0 * hits / TRIES;
31      System.out.println("Estimate for pi: " + piEstimate);
32  }
33 }
```

### Program Run

```
Estimate for pi: 3.1504
```



## Self Check 4.46

---

How do you simulate a coin toss with the `Math.random()` method?

**Answer:** Compute `(int) (Math.random() * 2)`, and use 0 for heads, 1 for tails, or the other way around.

## Self Check 4.47

---

How do you simulate the picking of a random playing card?

**Answer:** Compute `(int) (Math.random() * 4)` and associate the numbers 0 . . . 3 with the four suits. Then compute `(int) (Math.random() * 13)` and associate the numbers 0 . . . 12 with Jack, Ace, 2 . . . 10, Queen, and King.

## Self Check 4.48

---

Why does the loop body in `Dice.java` call `Math.random()` twice?

**Answer:** We need to call it once for each die. If we printed the same value twice, the die tosses would not be independent.

## Self Check 4.49

---

In many games, you throw a pair of dice to get a value between 2 and 12. What is wrong with this simulated throw of a pair of dice?

```
int sum = (int) (Math.random() * 11) + 2;
```

**Answer:** The call will produce a value between 2 and 12, but all values have the same probability. When throwing a pair of dice, the number 7 is six times as likely as the number 2. The correct formula is

```
int sum = (int) (Math.random() * 6) + (int)
(Math.random() * 6) + 2;
```

(1,1)	(1,6)
	(2,5)
	(3,4)
	(4,3)
	(5,2)
	(6,1)

## Self Check 4.50

---


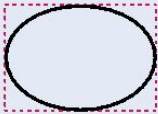
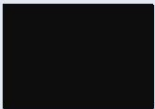

How do you generate a random floating-point number  $\geq 0$  and  $< 100$ ?

**Answer:** `Math.random() * 100.0`

# Drawing Graphical Shapes



---

Table 4 Graphics Methods

Method	Result	Notes
<code>g.drawRect(x, y, width, height)</code>		(x, y) is the top left corner.
<code>g.drawOval(x, y, width, height)</code>		(x, y) is the top left corner of the box that bounds the ellipse. To draw a circle, use the same value for width and height.
<code>g.fillRect(x, y, width, height)</code>		The rectangle is filled in.
<code>g.fillOval(x, y, width, height)</code>		The oval is filled in.

# Drawing Graphical Shapes

---

<code>g.drawLine(x1, y1, x2, y2)</code>		$(x1, y1)$ and $(x2, y2)$ are the endpoints.
<code>g.drawString("Message", x, y)</code>		$(x, y)$ is the basepoint.
<code>g.setColor(color)</code>	From now on, draw or fill methods will use this color.	Use <code>Color.RED</code> , <code>Color.GREEN</code> , <code>Color.BLUE</code> , and so on. (See Table 10.1 for a complete list of predefined colors.)

# TwoRowsOfSquares.java

```
11 public static void draw(Graphics g)
12 {
13     final int width = 20;
14     g.setColor(Color.BLUE);
15
16     // Top row. Note that the top left corner of the drawing has coordinates (0, 0)
17     int x = 0;
18     int y = 0;
19     for (int i = 0; i < 10; i++)
20     {
21         g.fillRect(x, y, width, width);
22         x = x + 2 * width;
23     }
24     // Second row, offset from the first one
25     x = width;
26     y = width;
27     for (int i = 0; i < 10; i++)
28     {
29         g.fillRect(x, y, width, width);
30         x = x + 2 * width;
31     }
32 }
```