Chapter 4

Flow of Control

- 4.1 Java Loop Statements
- 4.2 Programming with Loops
  - » exit(n) method
- 4.3 Graphics Supplement

4.1 JAVA LOOP STATEMENTS - Repetition: Loops

- Loop
  - » a portion of a program that repeats a statement or group of statements
- The body of loop
  - » the statement(or group of statements) to be repeated in a loop
- Iteration of a loop
  - » each repetition of the loop body

Loops

- Structure:
  - » ***Usually some initialization code***
  - » ***body of loop***
  - » ***loop termination condition***
- Several logical organizations
  - » 1) [redacted] loops
  - » 2) [redacted] loops
  - » infinite loops
  - » minimum of zero or minimum of one iteration
- Several programming statement variations
  - » ***while***
  - » ***do-while***
  - » ***for***

`while` Loop

- Syntax:

```
while(Boolean_Expression)
{
    //body of loop
    First_Statement;
    ...
    Last_Statement;
}
```

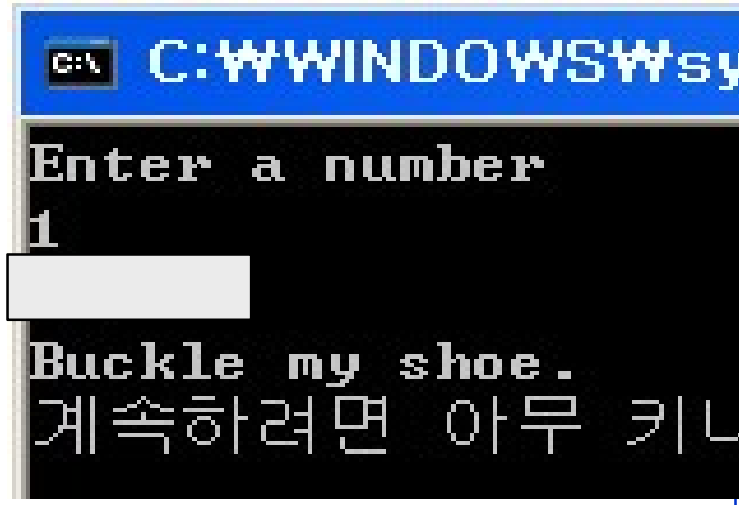
Something in body of loop should eventually cause *Boolean_Expression* to be *false*.

- Initialization statements usually precede the loop.
- *Boolean_Expression* is the loop termination condition.
- The loop will continue executing as long as *Boolean_Expression* is true.
- May be either counting or sentinel loop
  - » Good choice for sentinel loop

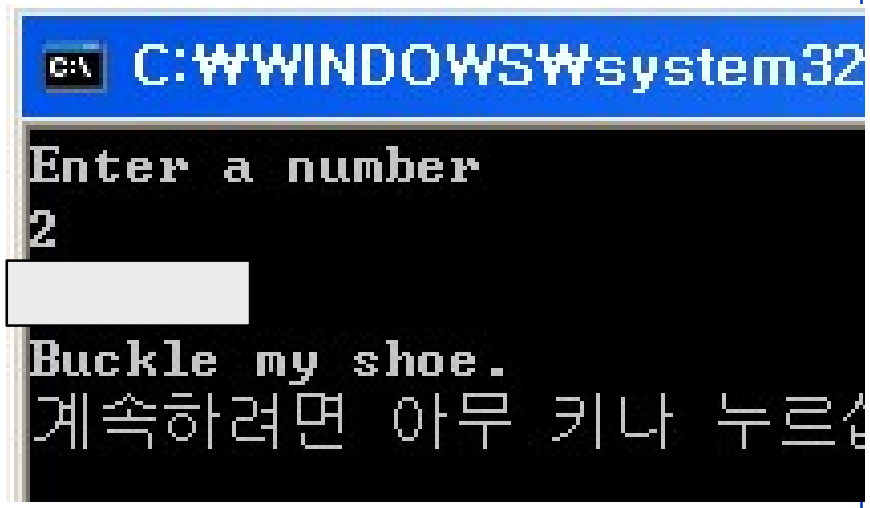
List 4.1

- List 4.1 A WHILE LOOP
  - » WhileDemo.java
  - » The Loop body is iterated zero times (when input is 0)

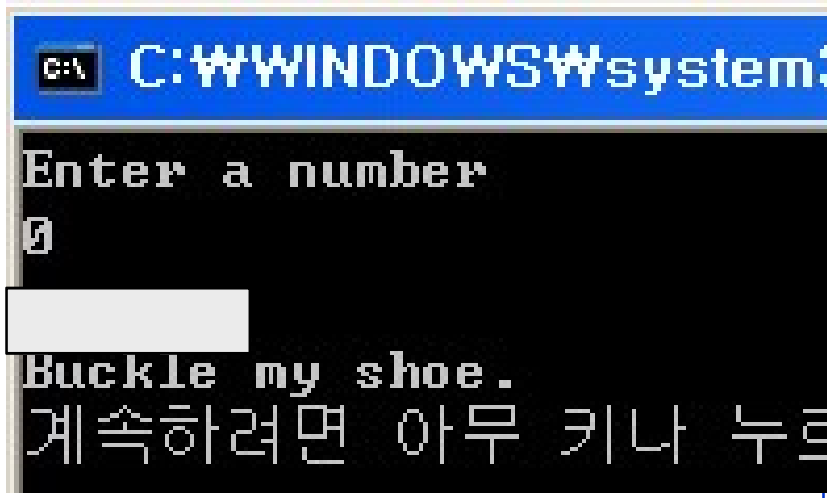
```
// List 4.1 A WHile Loop .

import java.util.Scanner;

public class WhileDemo
{
   public static void main(String[] args)
   {
      int count, number;
      System.out.println("Enter a number");
      Scanner keyboard = new Scanner(System.in); // 1, 2, 0
      number = keyboard.nextInt( );
      count = 1;
      while (count <= number)
      {
         System.out.print(count + ", ");
         count++;
      }
      System.out.println( );
      System.out.println("Buckle my shoe.");
   }
}
```



```
// List 4.1 A WHile Loop .

import java.util.Scanner;

public class WhileDemo
{
   public static void main(String[] args)
   {
      int count, number;
      System.out.println("Enter a number");
      Scanner keyboard = new Scanner(System.in); // 1, 2, 0
      number = keyboard.nextInt( );
      count = 1;
      while (count <= number)
      {
         System.out.print(count + ", ");
         count++;
      }
      System.out.println( );
      System.out.println("Buckle my shoe.");
   }
}
```

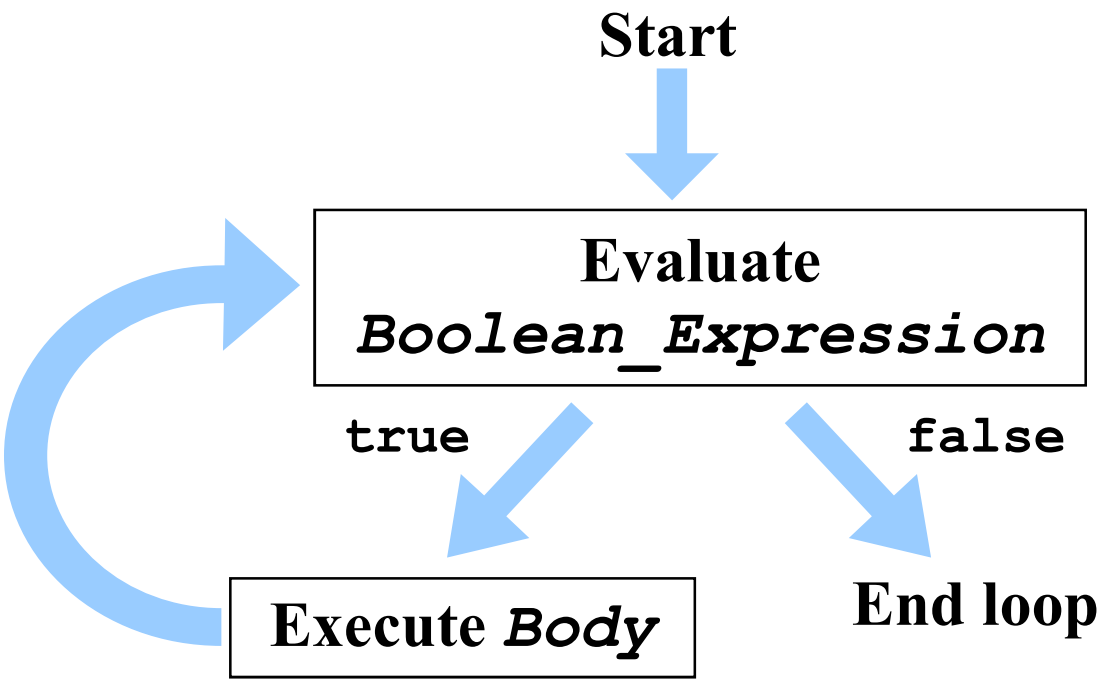


```
// List 4.1 A WHile Loop .

import java.util.Scanner;

public class WhileDemo
{
   public static void main(String[] args)
   {
      int count, number;
      System.out.println("Enter a number");
      Scanner keyboard = new Scanner(System.in); // 1, 2, 0
      number = keyboard.nextInt( );
      count = 1;
      while (count <= number)
      {
         System.out.print(count + ", ");
         count++;
      }
      System.out.println( );
      System.out.println("Buckle my shoe.");
   }
}
```



Semantics of the **`while`** Statement

```
while (Boolean_Expression)
        Body
```



`while`: ??

- 1) counting loop ??
- 2) sentinel controlled loop ??

`while`: A Counting Loop Example

- A loop to sum 10 numbers entered by user

```
int next;
//Loop initialization
int count = 1;
int total =0;
while(count <= 10) //Loop termination condition
{ //Body of loop
    next = keyboard.nextInt( );
    total = total + next;
    count++; //Loop termination counter
}
```

`while`: A Sentinel Controlled Loop Example

- A loop to sum positive integers entered by the user
- `next` is the sentinel
- The loop terminates when the user enters a negative number

```
//Initialization
int next = 0;
int total = 0;
while(next >= 0) //Termination condition
{ //Body
  total = total + next;
  next = keyboard.nextInt( );
}
```

`while`: A Minimum of Zero Iterations

- Because the first input value read and the test precedes the loop, the body the *`while`* loop body may not execute at all

```
//Initialization
int next;
int total = 0;
next = keyboard.nextInt( );
while(next >= 0)//Termination condition
{ //Body
  total = total + next;
  next = keyboard.nextInt( );
}
```

- If the first number the user enters is negative the loop body never executes

do-while Loop

- Syntax

```
do
{   //body of loop
    First_Statement;
    ...
    Last_Statement;
} while(Boolean_Expression);
```

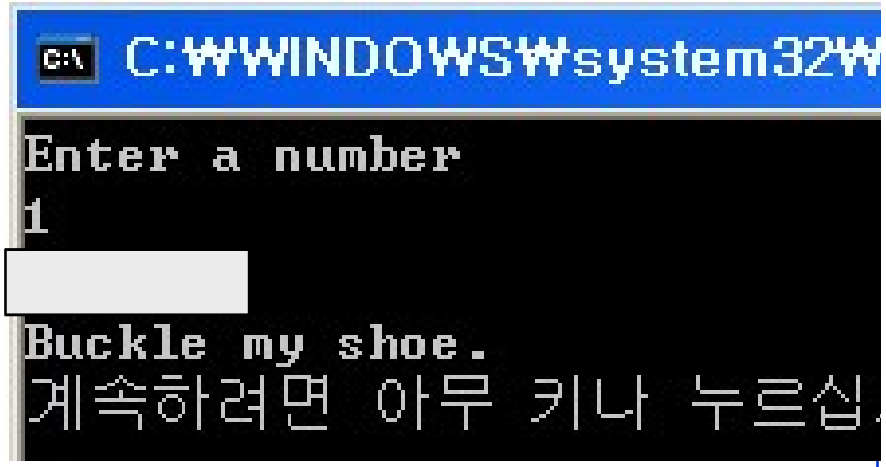
Something in body of loop should eventually cause *Boolean_Expression* to be *false*.

- Initialization code may precede loop body
- Loop test is after loop body so the body must execute at least once (minimum of at least one iteration)
- May be either counting or sentinel loop
  - » Good choice for sentinel loop

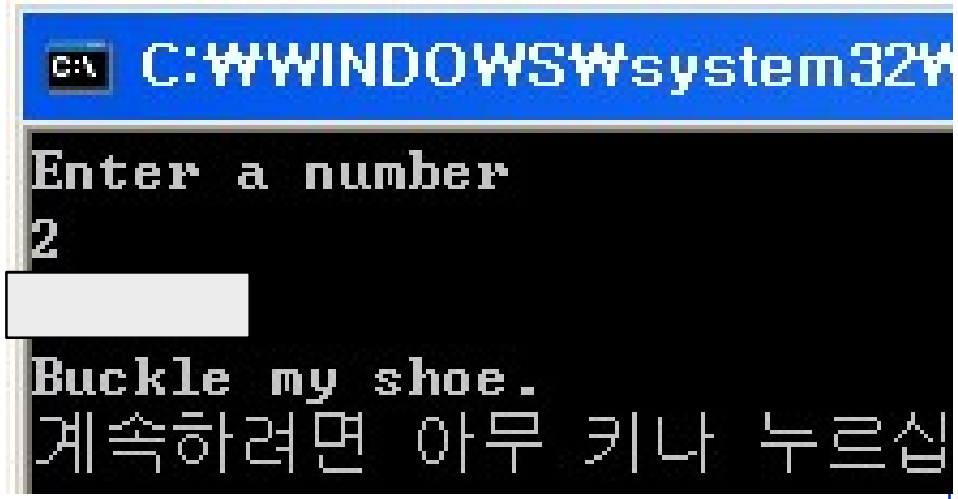
LISTING 4.2

- LISTING 4.2 A do-while Loop
  - »
  - » DOWhileDemo.java
  - » the loop body is always executed at least one time (although the input is 0)

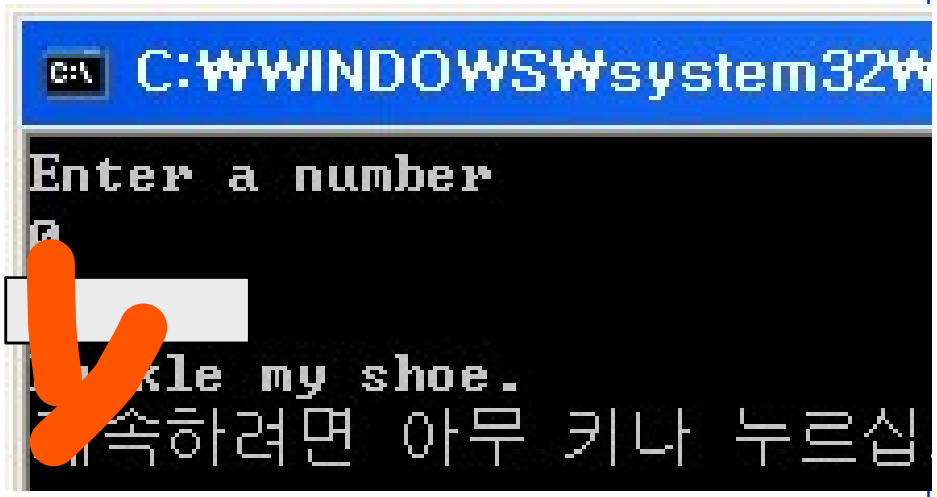
```
// Listing 4.2 A do-while Loop
import java.util.*;
public class DoWhileDemo
{
   public static void main(String[] args)
   {
      int count;
      int number;

      System.out.println("Enter a number");
      Scanner keyboard = new Scanner(System.in); // 1,2,0
      number = keyboard.nextInt( );
      count = 1;
      do
      {
         System.out.print(count + ", ");
         count++;
      }while (count <= number);
      System.out.println( );
      System.out.println("Buckle my shoe.");
   }
}
```



```
// Listing 4.2 A do-while Loop
import java.util.*;
public class DoWhileDemo
{
   public static void main(String[] args)
   {
      int count;
      int number;

      System.out.println("Enter a number");
      Scanner keyboard = new Scanner(System.in); // 1,2,0
      number = keyboard.nextInt( );
      count = 1;
      do
      {
         System.out.print(count + ", ");
         count++;
      }while (count <= number);
      System.out.println( );
      System.out.println("Buckle my shoe.");
   }
}
```

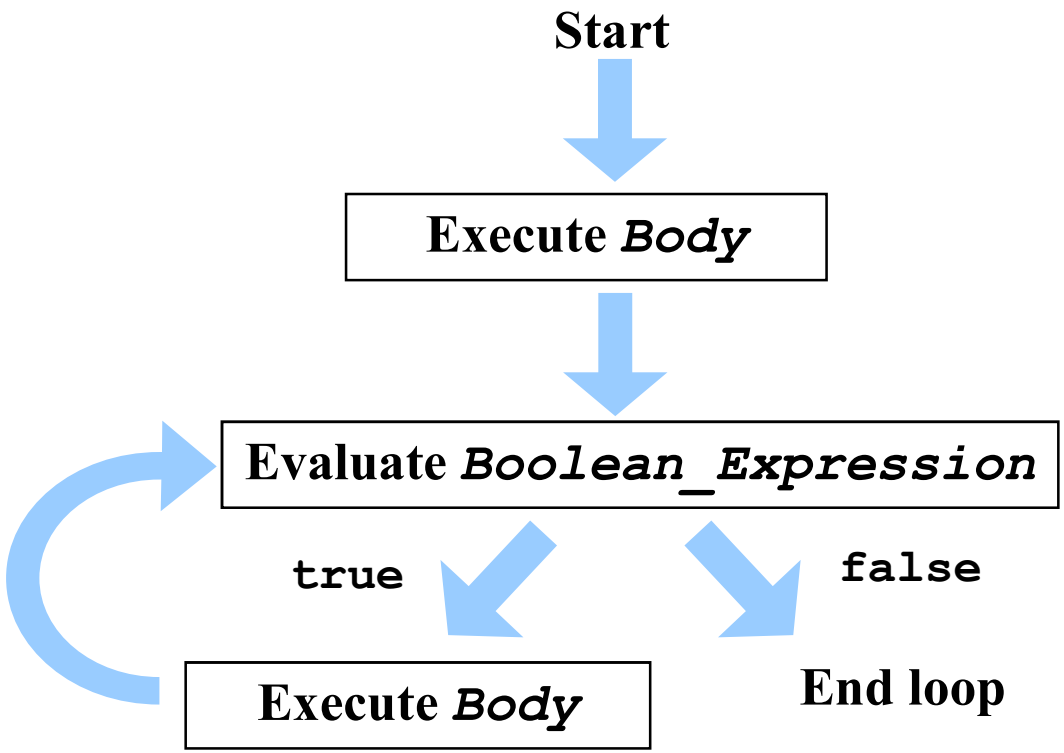


```
// Listing 4.2 A do-while Loop
import java.util.*;
public class DoWhileDemo
{
    public static void main(String[] args)
    {
        int count;
        int number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in); // 1,2,0
        number = keyboard.nextInt( );
        count = 1;
        do
        {
            System.out.print(count + ", ");
            count++;
        }while (count <= number);
        System.out.println( );
        System.out.println("Buckle my shoe.");
    }
}
```



Semantics of the **do-while** Statement

```
do
    Body
While (Boolean_Expression);
```



do-while Example

```
int count = 1;
int number = 5;
do //Display integers 1 to 5 on one line
{
   System.out.print(count + " ");
   count++;
} while(count <= number);    //output??
```

Note that `System.out.print()` is used and not `System.out.println()` so the numbers will all be on one line.

Output:

LISTING 4.3

- LISTING 4.3 Roach Population Program

```
// LISTING 4.3
import java.util.Scanner;
/**
 Program to calculate how long it will take a population of
 roaches to completely fill a house from floor to ceiling.
*/
public class BugTrouble
{
   public static final double GROWTH_RATE = 0.95;     //95% per week
   public static final double ONE_BUG_VOLUME = 0.002; //cubic feet

   public static void main(String[] args)
   {
      System.out.println("Enter the total volume of your house");
      System.out.print("in cubic feet: ");
      Scanner keyboard = new Scanner(System.in);
      double houseVolume = keyboard.nextDouble( );
```

```
System.out.println("Enter the estimated number of");
      System.out.print("roaches in your house: ");
      int startPopulation = keyboard.nextInt( );
      int countWeeks = 0;
      double population = startPopulation;
      double totalBugVolume = population * ONE_BUG_VOLUME;

      while (totalBugVolume < houseVolume)
      {
                                    double newBugs = population * GROWTH_RATE;
                                    double newBugVolume = newBugs * ONE_BUG_VOLUME;
                                    population = population + newBugs;
                                    totalBugVolume = totalBugVolume + newBugVolume;
         countWeeks++;
      }

      System.out.println("Starting with a roach population of " +
                  startPopulation);
      System.out.println("and a house with a volume of " + houseVolume +
                  " cubic feet,");
      System.out.println("after " + countWeeks + " weeks,");
      System.out.println("the house will be filled with " +
                                                            (int)population + " roaches.");
      System.out.println("They will fill a volume of " +
                   (int)totalBugVolume + " cubic feet.");

      System.out.println("Better call Debugging Experts Inc.");
   }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter the total volume of your house
in cubic feet: 20000
Enter the estimated number of
roaches in your house:100
Starting with a roach population of 100
and a house with a volume of 20000.0 cubic feet,
after 18 weeks,
the house will be filled
floor to ceiling with roaches.
There will be 16619693 roaches.
They will fill a volume of 33239 cubic feet.
Better call Debugging Experts Inc.
계속하려면 아무 키나 누르십시오 . . .
```

Infinite loop

- A loop that iterates its body repeatedly without ever ending

```
public class BugTrouble
{
    public static final double GROWTH_RATE =-0.05;
    public static final double ONE_BUG_VOLUME = 0.002;//cubic feet
    public static void main(String[] args)
    {
        ......
        while (totalBugVolume < houseVolume)
        {
            population = population + (GROWTH_RATE*population);
            totalBugVolume = population*ONE_BUG_VOLUME;
            countWeeks++;
        } .............
}
```

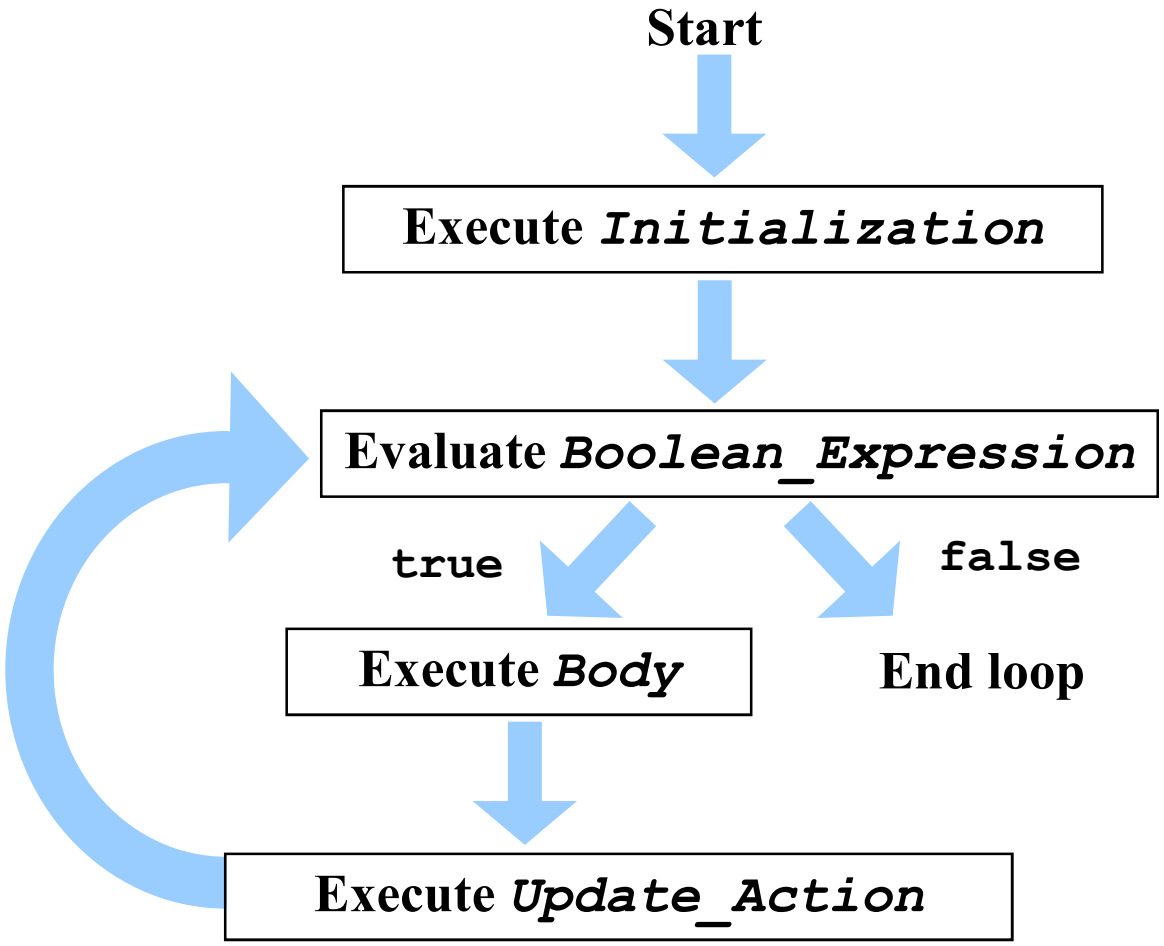
`for` Loop

- Good choice for counting loop
- Initialization, loop test, and loop counter change are part of the syntax
- Syntax:

```
for(Initialization; Boolean_Expression;
Update_Action)
   loop body;
```

Semantics of the **for** Statement

```
for(Initialization; Boolean_Expression;
Update_Action)
    loop body;
```



For loop && while loop

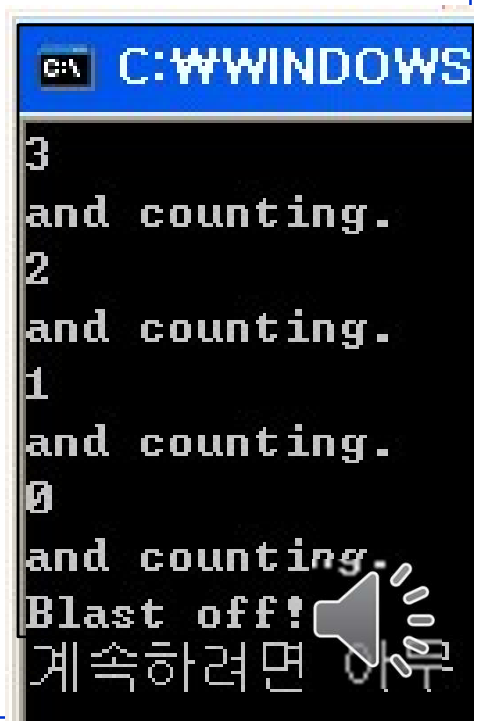
```
For (initializing_Action;
Boolean_Expression;
Update_Action)
{
    First_Statement
    Second_Statement
    ....
    ....
    ....
    ....
    Last_Statement
}
```

```
While (
{
    First_Statement
    Second_Statement
    ....
    ....
    ....
    ....
}
```

LISTING 4.5

- LISTING 4.5 A for Statement

```
// LISTING 4.5 A for Statement
// Screen Output
public class ForDemo
{
    public static void main(String[] args)
    {
        int countDown;
        for (countDown = 3; countDown >= 0; countDown--)
        {
            System.out.println(countDown);
            System.out.println("and counting.");
        }
        System.out.println("Blast off!");
    }
}
```



The comma in for Statements

- A for loop can perform **more than one initialization**
- **Multiple update actions** by stringing them together with commas.

```
for (n=1, product=1; n<=10; n++)
    product = product * n ;
```

```
for (                                    );
```

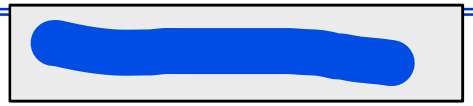
Extra Semicolon in a Loop Statement

```
int product = 1, number;
for (number=1; number<=10; number++);
    product = product * number ;
System.out.println(
    "Product of the numbers 1 through 10 is    " +  product);
```

```
Product of the numbers 1 through 10 is
1*2*3*4*5*6*7*8*9*10
 for (number=1; number<=10; number++)
 {
    // Do nothing
 }
```

Extra Semicolon in a Loop Statement

```
int product = 1, number=1;
while (number<=10);
{
    product = product * number ;
    number++;
}
System.out.println(
    "Product of the numbers 1 through 10 is    " +  product);
```



Choosing a Loop Statement

- for all possible inputs to your program, the loop should be iterated at least one time ➔ [redacted] statement
- a loop requires the possibility of iteating the body zero times ➔ [redacted] statement, [redacted] statement
- If it is a computation that changes some numeric quantity by some equal amount on each iteration ➔ [redacted] statement
- The while statement is always the safest choice.

The break statement in Loops

- If the loop is contained within a larger loop(or if the loop is inside a switch statement), then the break statement ends only the innermost loop.
- LISTING . Ending a Loop with a break statement
  - » BreakDemo.java

```
// LISTING Ending a loop with a break statement

public class BreakDemo
{
   public static void main(String[] args)
   {
      int itemNumber;
      double amount, total;

      System.out.println("You may buy ten items, but");
      System.out.println("the total price must not exceed $100.");
```

```
        total = 0;
        for (itemNumber = 1; itemNumber <= 10; itemNumber++)
        {
            System.out.print("Enter cost of item #"
                                + itemNumber + ": $");
            amount = Scanner.nextInt();
            total = total + amount;
            if (total >= 100)
            {
                System.out.println("You spent all your money.");
                break;  // break
            }
            System.out.println("Your total so far is $" + total);
            System.out.println("You may purchase up to "
                    + (10 - itemNumber) + " more items.");
        }

        System.out.println("You spent $" + total);
    }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
You may buy ten items, but
the total price must not exceed $100.
Enter cost of item #1: $90.93
Your total so far is $90.93
You may purchase up to 9 more items.
Enter cost of item #2: $10.50
You spent all your money.
You spent $101.43
계속하려면 아무 키나 누르십시오 . .
```

Misuse of break statements

- A loop without a break statement has a simple, easy-to-understand structure
- A break statement make it more difficult to understand the loop
- They should be used at most sparingly.

Nested Loops

- The body of a loop can have any kind of statements, including another loop.

```
for (line = 0; line < 4; line++)
{
    for (star = 0; star < 5; star++)
        System.out.print('*');
    System.out.println();}
```

body of outer loop

body of inner loop

- Each time the outer loop body is executed, the inner loop body will execute 5 times, making a total of 20 times.

???Output:

LISTING 4.4

```
// LISTING 4.4 Nested Loops

import java.util.*;

/**
 Determines the average of a list of (nonnegative) exam scores.
 Repeats for more exams until the user says to stop.
*/
public class ExamAverager
{
   public static void main(String[] args)
   {
      System.out.println("This program computes the average of");
      System.out.println("a list of (nonnegative) exam scores.");
      double sum;
      int numberOfStudents;
      double next;
      String answer;
      Scanner keyboard = new Scanner(System.in);
```

```
do
    {
        System.out.println( );
        System.out.println("Enter all the scores to be averaged.");
        System.out.println("Enter a negative number after");
        System.out.println("you have entered all the scores.");
        sum = 0;
        numberOfStudents = 0;
        next = keyboard.nextDouble( );
        while (next >= 0)
        {
            sum = sum + next;
            numberOfStudents++;
            next = keyboard.nextDouble( );
        }
        if (numberOfStudents > 0)
            System.out.println("The average is "
                                + (sum/numberOfStudents));
       else
            System.out.println("No scores to average.");
        System.out.println("Want to average another exam?");
        System.out.println("Enter yes or no.");
        answer = keyboard.next( );
    }while (answer.equalsIgnoreCase("yes"));

  }
}
```

```
C:₩Windows₩system32₩cmd.exe
This program computes the average of
a list of (nonnegative) exam scores.

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
100
90
100
90
-1
The average is 95.0
Want to average another exam?
Enter yes or no.
yes

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
90
80
70
-1
The average is 80.0
Want to average another exam?
Enter yes or no.
no
계속하려면 아무 키나 누르십시오 . . .
```

끝