LOOP

Motivation

Suppose that you need to print a string (e.g., "Welcome to Java!") a thousand times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to
Java!");
```

So, how do you solve this problem?

Motivation

Problem:

1000 times

```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");

…
…
…
…
…

System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

Motivation

A solution using *While* Loop:

```
int count = 0;
while (count < 1000)
{
  System.out.println("Welcome to Java!");
  count++;
}
```

A solution using *for* Loop:

```
for (int count=1; count <= 1000; count=count+1)
  System.out.println("Welcome to Java!");
```

1. Loop Statements

- *Loops are repetition statements* that allow us to execute a statement (or block of statements) multiple times
- Like conditional statements, they are controlled by boolean expressions
- Java has three types of loop statements:
  - the *while loop*
  - the *do-while loop*
  - the *for loop*
- The programmer should choose the right type of loop for the situation at hand

Loop Statements

- The *while and do-while* loops are also called conditional loops since they use <u>boolean expressions</u> to control the loop behavior
- The *while and do-while* loops run un-determined (unknown) number of iterations (some call them <u>non-deterministic</u> loops)
- The *for* loop, on the other hand, runs a pre-determined (known) number of iterations (some call it <u>deterministic</u> loop or <u>counting</u> loop)

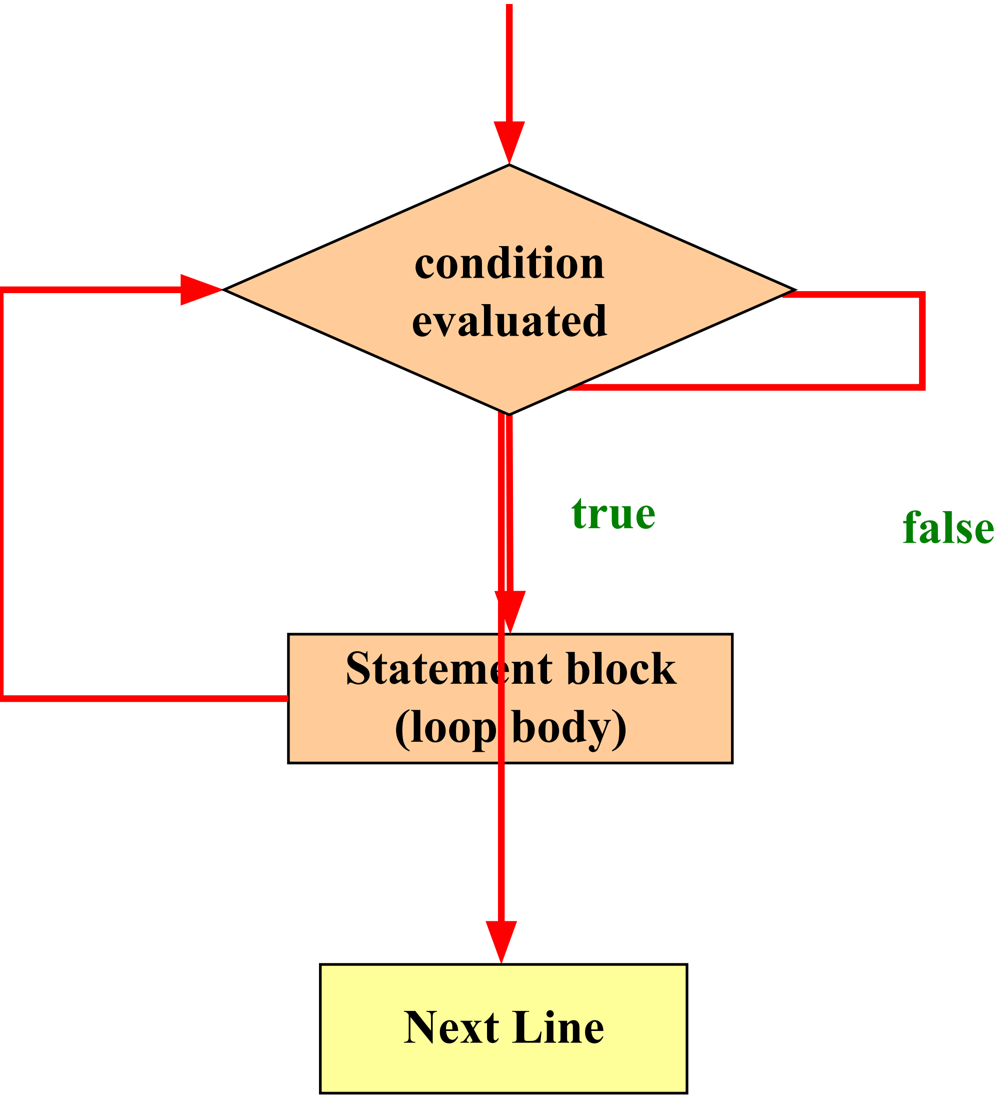
2. *while* Loop Statement

- A *while loop (statement)* has the following syntax:

```
while (condition)
        statement block; //loop body
```

- **If the `condition` is true, the `statement block` is executed**
- **Then the condition is evaluated again, and if it is still *true*, the statement is executed again**
- **The statement is executed repeatedly until the condition becomes *false***

while Loop Logic



<u>Note:</u> If the initial evaluation of the condition is false, the loop body executes <u>zero times</u>. Therefore, the while loop executes zero or more times

Trace while Loop

Initialize count

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

Trace while Loop, cont.

(count < 2) is true

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

Trace while Loop, cont.

Print Welcome to Java

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

Trace while Loop, cont.

Increase count by 1
count is 1 now

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

Trace while Loop, cont.

(count < 2) is still true since count is 1

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

Trace while Loop, cont.

Print Welcome to Java

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

Trace while Loop, cont.

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

Increase count by 1
count is 2 now

Trace while Loop, cont.

(count < 2) is false since count is 2 now

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

Trace while Loop

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

The loop exits. Execute the next statement after the loop.

while Loop Example

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
   System.out.println (count);
   count = count + 1;
}
```

- **If the condition is false initially, the statement (loop body) is <u>never executed</u>**
- **Therefore, the body of a `while` loop will execute zero or more times**

while Loop Sentinel Value

Question: How can we control a while loop?

- A *sentinel value* is a special input value that represents the end of inputs from the user
- The *sentinel value* should be included in the prompt so that the user knows how to stop the loop. For example,

```
System.out.println("Enter a grade (type 9999 to quit): ");
```

- A *sentinel value* gives the user control over the loop
- See Average.java next slide

Sentinel Value Example

```
// Demonstrates the use of a while loop using a sentinel value
import java.text.DecimalFormat;
import java.util.Scanner;
public class Average
{
  public static void main (String[] args)
  { int sum = 0, value, count = 0;
    double average;
    Scanner scan = new Scanner (System.in);
    System.out.print ("Enter an integer (0 to quit): ");
    value = scan.nextInt();
    while (value != 0)  //sentinel value of 0 to terminate loop
    { count = count + 1;
      sum = sum + value;
      System.out.println ("The sum so far is " + sum);
      System.out.print ("Enter an integer (0 to quit): ");
      value = scan.nextInt();
    }
    System.out.println ();
    if (count == 0)
        System.out.println ("No values were entered.");
    else
       System.out.println ("Sum of all values = " + sum);
   }
}
```

while Loops for Input Validation

- A while loop can be used for *input validation*, making a program more *robust*
- Input validation allows the program to ensure correct input values before the input is processed
- It also allows the program to issue error messages to the user when invalid data is entered
- See WinPercentage.java next slide

Input Validation Example

```java
// Demonstrates the use of a while loop for input validation
import java.text.NumberFormat;
import java.util.Scanner;
public class WinPercentage
{
  public static void main (String[] args)
   {
     final int NUM_GAMES = 12;
     int won;
     double ratio;
     Scanner scan = new Scanner (System.in);
     System.out.print ("Enter the number of games won (0 to "
                         + NUM_GAMES + "): ");
     won = scan.nextInt();

     //input validation
     while (won < 0 || won > NUM_GAMES)
     {
        System.out.print ("Invalid input. Please reenter: ");
        won = scan.nextInt();
     }

     ratio = (double)won / NUM_GAMES;
     NumberFormat fmt = NumberFormat.getPercentInstance();
     System.out.println ();
     System.out.println ("Winning percentage: " +fmt.format(ratio));
 }
}
```

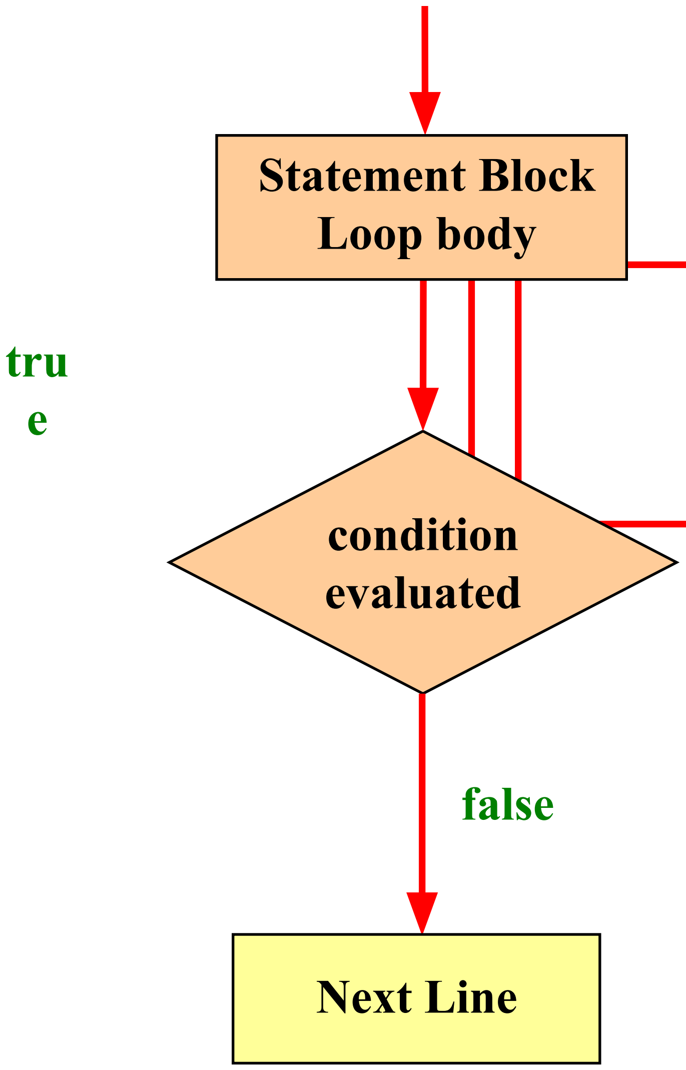
3. *do-while* Loop

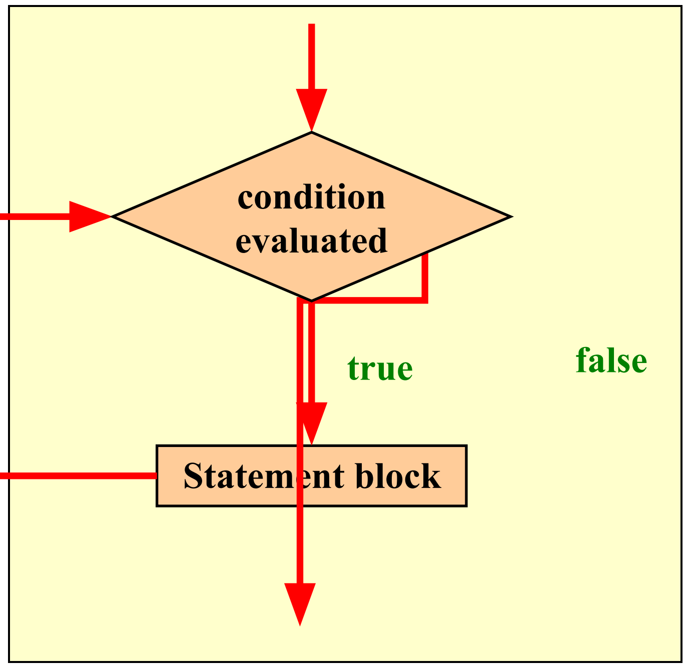
- A *do-while loop* has the following syntax:

```
do
{
   statement block;
} while (condition)
```

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

Logic of *do-while* Loop





While Loop

do-while Loop Example

- An example of a `do` loop:

```
int count = 0;
do
{
   count = count +1;
   System.out.println (count);
} while (count < 5);
```

- **The body of a `do` loop executes at least once**
- **See [ReverseNumber.java] next slide**

do-while Loop Example

```
// Demonstrates the use of a do loop
import java.util.Scanner;
public class ReverseNumber
{
  public static void main (String[] args)
  {
    int number, lastDigit, reverse = 0;
    Scanner scan = new Scanner (System.in);
    System.out.print ("Enter a positive integer: ");
    number = scan.nextInt();

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

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

4. *for* Loop

- A *for statement* has the following syntax:

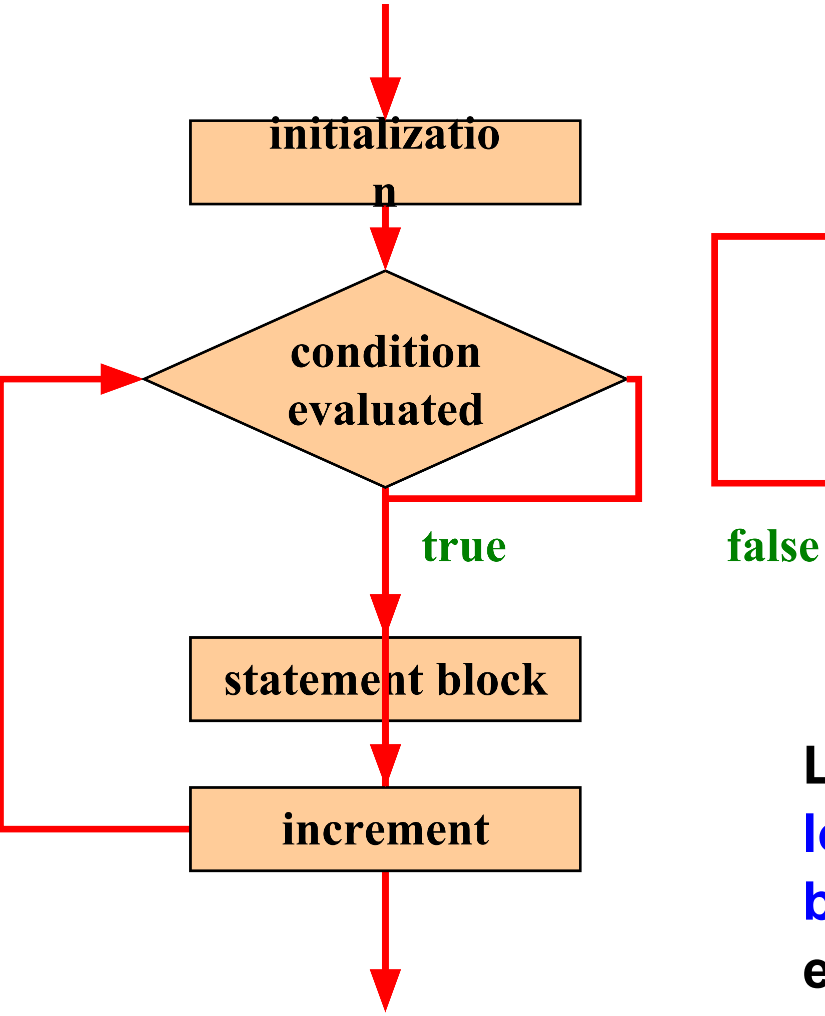
The *`initialization`* is <u>executed once</u> before the loop begins

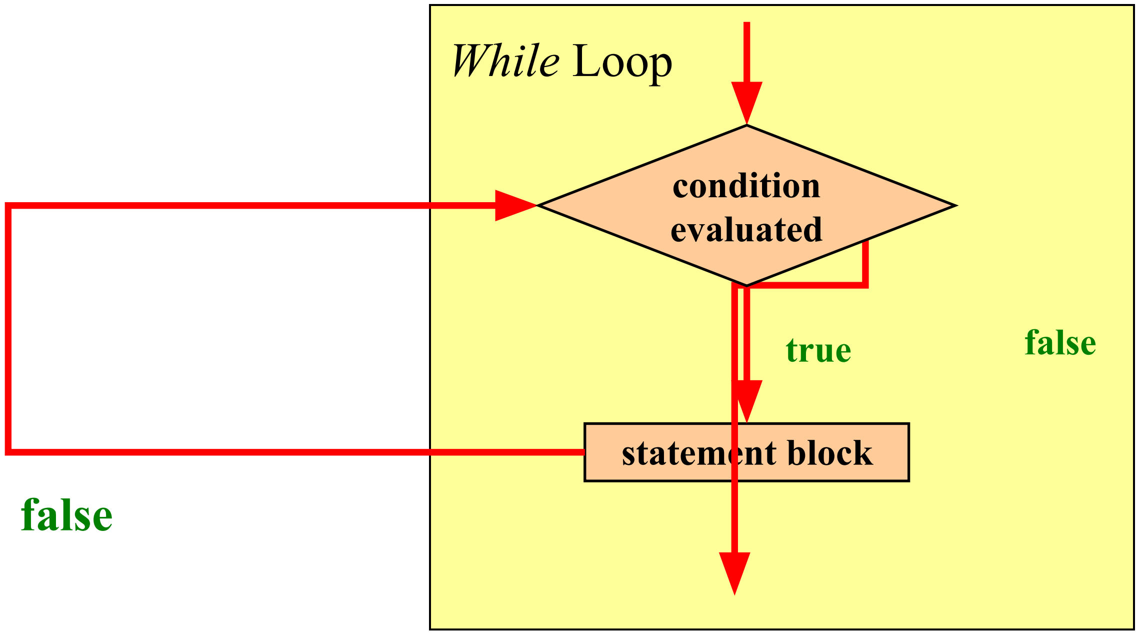
The *`statement`* is executed until the *`condition`* becomes false

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

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

for Loop Logic





Like a *while* loop, the condition of a *for* loop is tested prior to executing the loop body. Therefore, the *for* loop body will execute zero or more times

Trace for Loop

Declare i

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

Trace for Loop, cont.

Execute initializer
i is now 0

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

Trace for Loop, cont.

(i < 2) is true
since i is 0

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

Trace for Loop, cont.

Print Welcome to Java

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

Trace for Loop, cont.

Execute adjustment statement
i now is 1

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

Trace for Loop, cont.

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

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

Trace for Loop, cont.

Print Welcome to Java

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

Trace for Loop, cont.

Execute adjustment statement
i now is 2

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

Trace for Loop, cont.

(i < 2) is false
since i is 2

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

Trace for Loop, cont.

Exit the loop. Execute the next statement after the loop

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

for Loop as a *while* Loop

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

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

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

for to *while* Loop Example

- The `for` loop:

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

  - **The initialization section can be used to declare a variable, making it is local valuable to the loop body.**

```
int count = 1;
while (count <= 5)
{
  System.out.println (count);
  count = count + 1;
}
```

for Loop Example

- The increment section can perform any calculation

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

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

for Loop Example

```
// Demonstrates the use of a for loop to print multiples of a number
import java.util.Scanner;
public class Multiples
{
  public static void main (String[] args)
  {
   final int PER_LINE = 5;
   int value, limit, multiple, count = 0;
   Scanner scan = new Scanner (System.in);
   System.out.print ("Enter a positive value: ");
   value = scan.nextInt();
   System.out.print ("Enter an upper limit: ");
   limit = scan.nextInt();
   System.out.println ();
   System.out.println ("The multiples of " + value + " between " +
                   value + " and " + limit + " (inclusive) are:");
   for (multiple = value; multiple <= limit; multiple = multiple + value)
   {
     System.out.print (multiple + "\t");
     // Print a specific number of values per line of output
     count = count + 1;
     if (count % PER_LINE == 0)
        System.out.println(); // go to next line
   }
  }
}
```

5. Infinite Loops

- The body of a `while` loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally

Example

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
   System.out.println (count);
   count = count - 1; //Error
}
```

  - **This loop will continue executing until interrupted (Control-C) or until an underflow error occurs**

Be Careful!

- If the *condition* is left out, it is always considered to be true, and therefore creates an infinite loop

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

- If the *increment* is left out, no increment operation is performed, and therefore creates an infinite loop

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

Equivalent

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

6. Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain other loop statements
- For each iteration of the outer loop, the inner loop iterates completely
- See PalindromeTester.java next slide

Example

```
// Demonstrates the use of nested while loops.
import java.util.Scanner;
public class PalindromeTester
{
  public static void main (String[] args)
  { String str, another = "y";
    int left, right;
    Scanner scan = new Scanner (System.in);
    while (another.equalsIgnoreCase("y")) // allows y or Y
    {
       System.out.println ("Enter a potential palindrome string:");
       str = scan.nextLine();
       left = 0;
       right = str.length() - 1;
       while (str.charAt(left) == str.charAt(right) && left < right)
       {
          left = left + 1;
          right = right - 1;
       }
       System.out.println();
       if (left < right)
          System.out.println ("That string is NOT a palindrome.");
       else
          System.out.println ("That string IS a palindrome.");
       System.out.println();
       System.out.print ("Test another palindrome (y/n)? ");
       another = scan.nextLine();
    }
  }
}
```

Example

```
// Demonstrates the use of nested for loops to print starts
public class Stars
{
   public static void main (String[] args)
   {
      final int MAX_ROWS = 10;

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


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

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

Nested Loops Iterations

How many times will the string "I am here" be printed?

```
// Demonstrates the use of nested loops
public class NestedLoops
{
   public static void main (String[] args)
   { String str, another = "y";

      int count1 = 1;
      while (count1 <= 10)
      {
         int count2 = 1;
         while (count2 <= 5)
         {
            System.out.println("I am here!");
            count2 = count2 + 1;
         }
         System.out.println(); // blank line
        count1 = count1 + 1;
      }
   }
}
```

7. Using `break` and `continue`

Examples for using the `break` statement:

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

      while (number < 20)
      {
         number = number + 1;
         sum = sum + number;
         if (sum >= 100)   // stop if sum is over 100
            break;
      }

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

Using `break` and `continue`

Examples for using the `continue` statement:

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

      while (number < 10) {
         number = number + 1;
         if (number == 5 || number == 6)
            continue;  // do not add 5 and 6 to sum
         sum = sum + number;
      }

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

   }
}
```