Chapter 3 Control Statements

Comparison operators

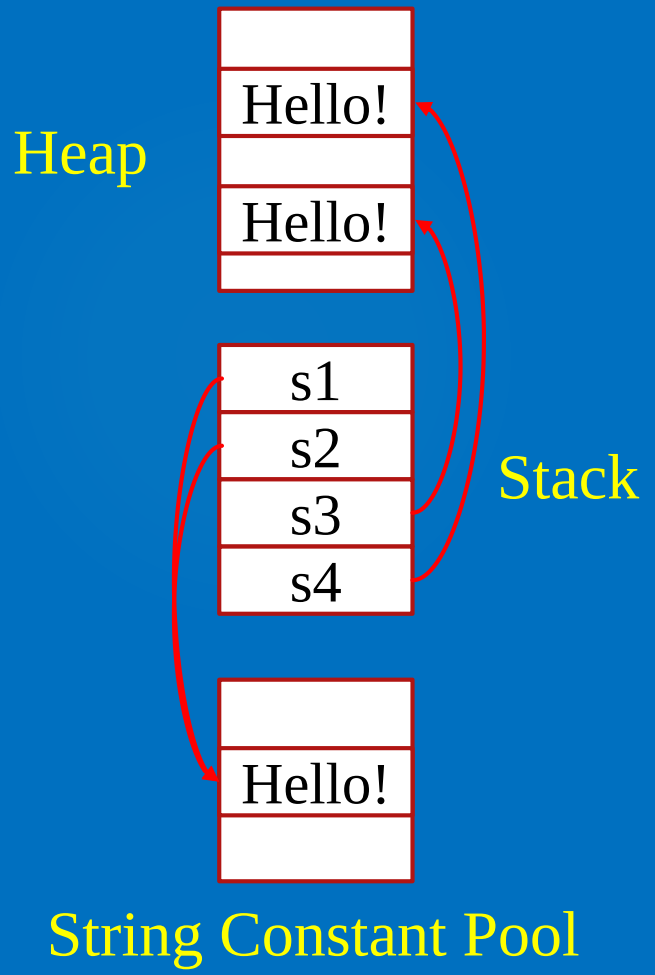
Operator	Name
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Comparison for primitive data types


```
public class TestClass {
    public static void main(String[] args) {
        int i1 = 30;
        int i2 = 20;
        double d1 = 20.0;
        System.out.println(i1 > i2); //true
        System.out.println(i1 < i2); //false
        System.out.println(i1 == i2); //false
        System.out.println(i1 != i2); //true
        System.out.println(i1 == d1); //false
    }
}
```

Comparison for String: string1 == string2 ?


```
public class TestClass {
    public static void main(String[] args) {
        String s1 = "Hello!";
        String s2 = "Hello!";
        String s3 = new String("Hello!");
        String s4 = new String("Hello!");
        System.out.println(s1==s2); //true
        System.out.println(s1==s3); //false
        System.out.println(s3==s4); //false
    }
}
```

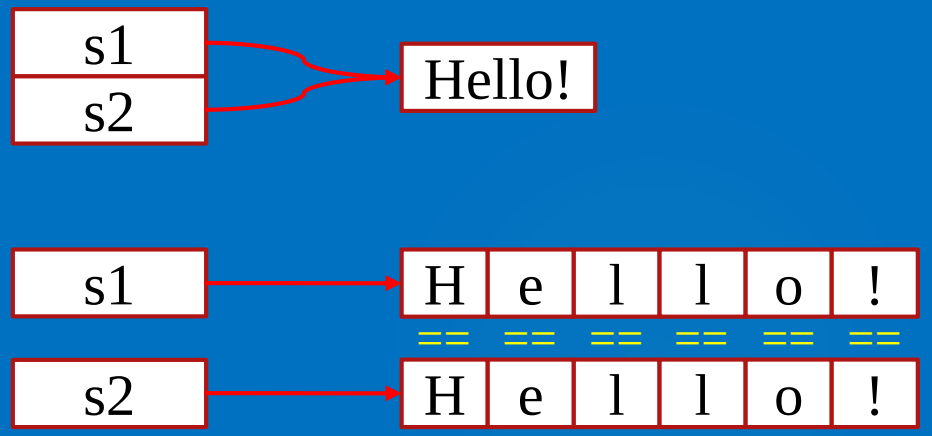


Comparison Method: equals()

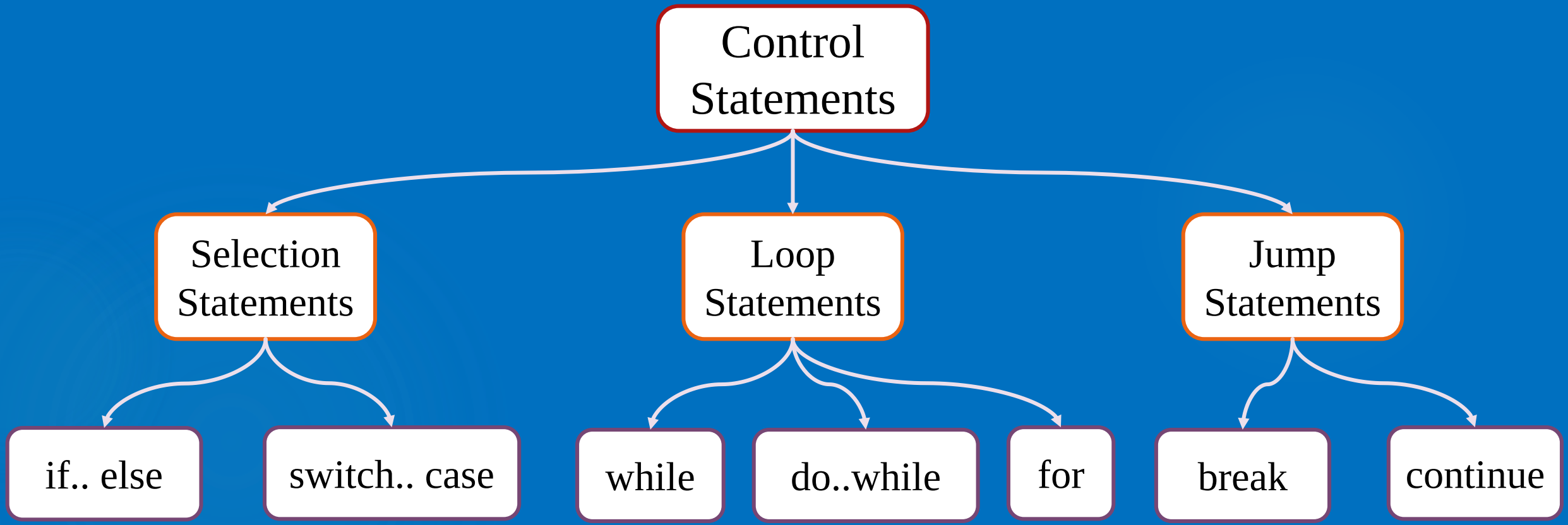

```
public class TestClass {
    public static void main(String[] args) {
        String s1 = "Hello!";
        String s2 = "Hello!";
        String s3 = new String("Hello!");
        String s4 = new String("Hello!");
        System.out.println(s1.equals(s2)); //true
        System.out.println(s1.equals(s3)); //true
        System.out.println(s1.equals(s4)); //true
    }
}
```

equals() in String class

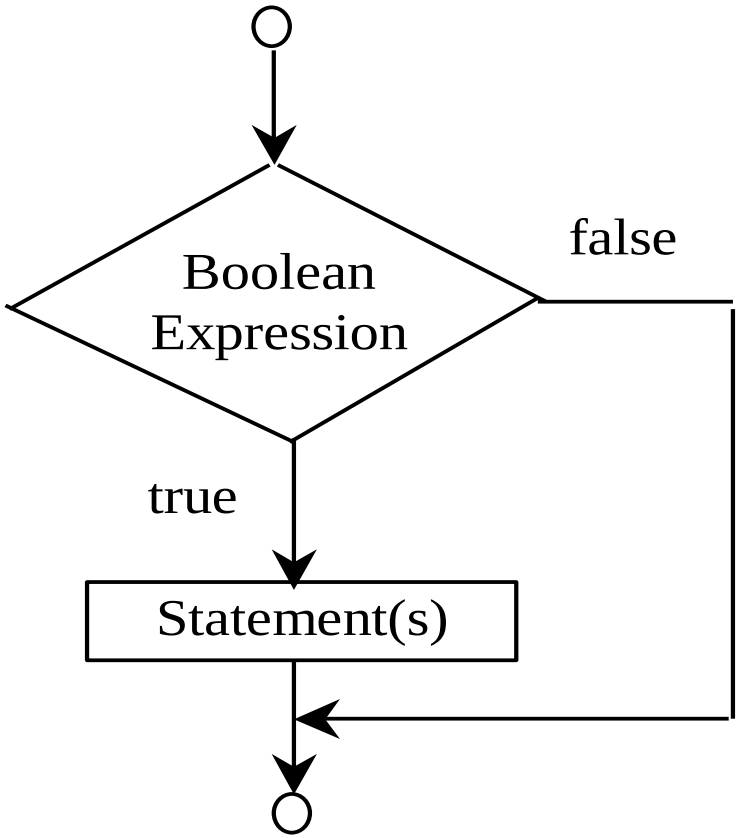

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
    return false;
}
```



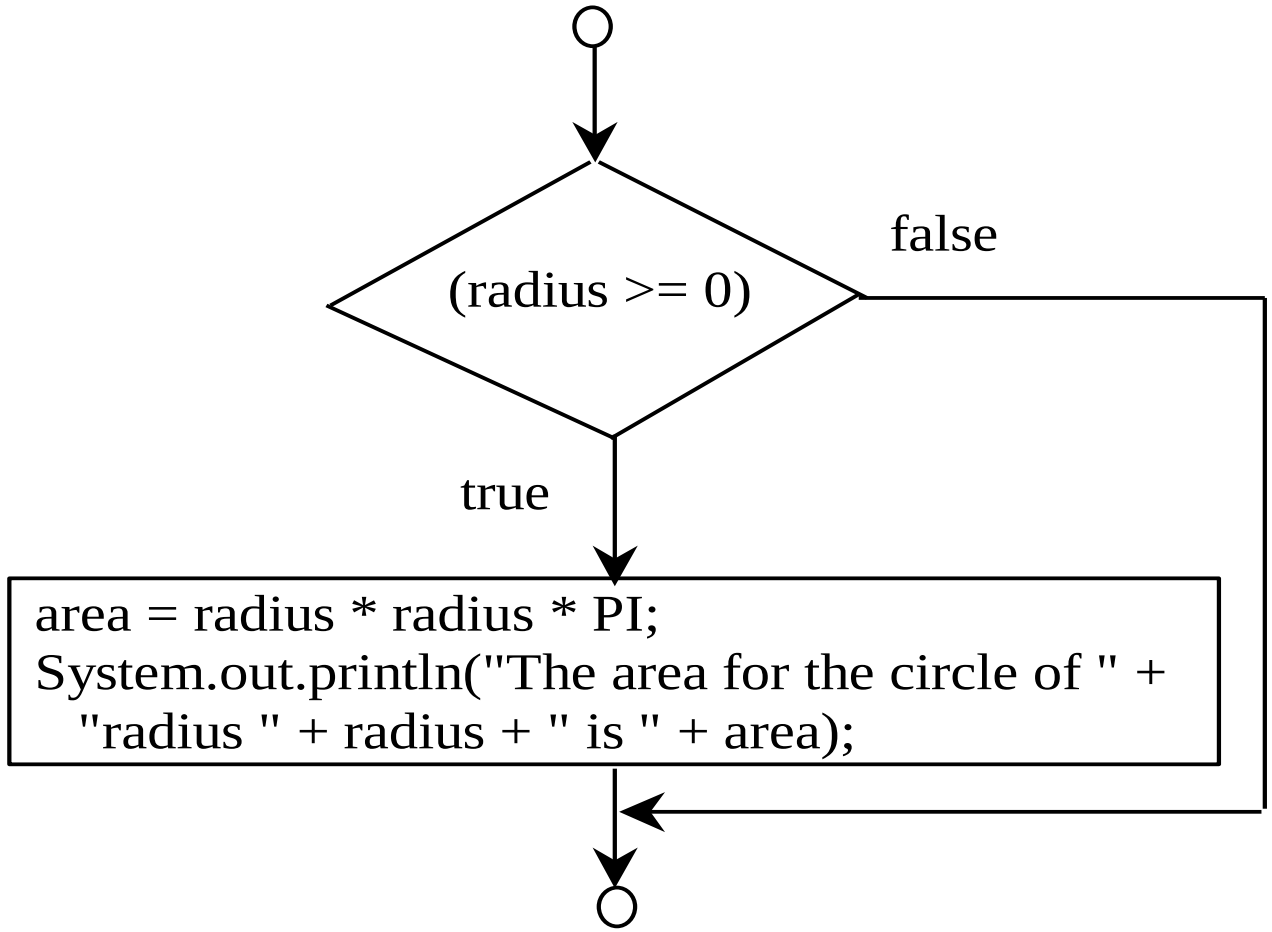
Java Control Statements



One-way if Statements



(A)



(B)

One-way if Statements


```
/*
    if (boolean-expression){
        statement(s);
    }
*/
public class TestClass {
    public static void main(String[] args){
        double radius = 2.0;
        if (radius >= 0) {
            double area = radius * radius * Math.PI;
            System.out.printf("The area is %4.2f", area);
        }
    }
}
```

Note


```
if i > 0 {
   System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
   System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
   System.out.println("i is positive");
}
```

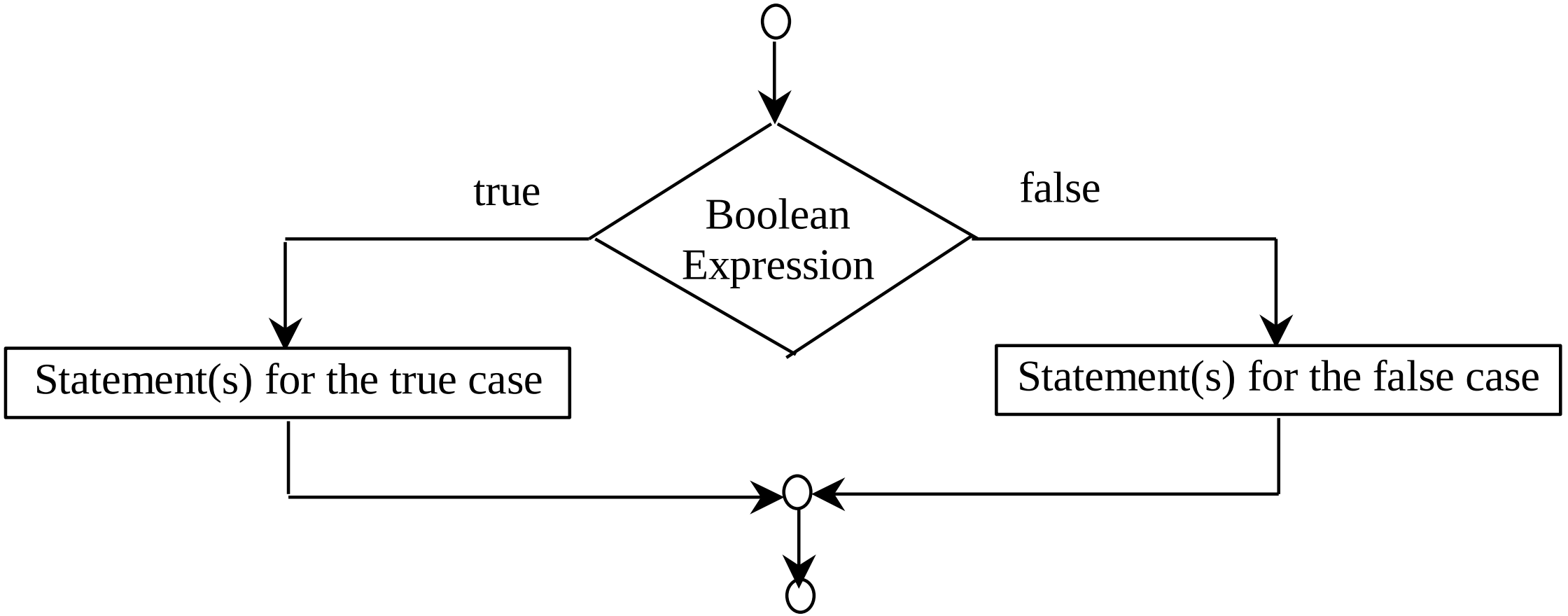
(a)

Equivalent

```
if (i > 0)
   System.out.println("i is positive");
```

(b)

The Two-way if Statement



The Two-way if Statement


```
/*  if (boolean-expression) {
        statement(s)-for-the-true-case;
    } else {
        statement(s)-for-the-false-case;
    }  */
public static void main(String[] args){
    double radius = 2.0;
    if (radius >= 0) {
        double area = radius * radius * Math.PI;
        System.out.printf("The area is %4.2f", area);
    } else{
        System.out.println("The radius is invalid");
    }
}
```

Multiple Alternative if Statements


```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Note: if-else match

The *else* clause matches *the most recent if* clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

Equivalent

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

To force the *else* clause to match the first *if* clause, you must add a pair of *braces*.

Common Errors


```
if (radius >= 0)
    String str = "Hi!";
    System.out.println(str);
```

Forgetting Braces

```
if (radius >= 0);
    String str = "Hi!";

if (radius >= 0){};
```

Wrong Semicolon at if Line

```
boolean isStudent = true;
if (isStudent == true){
    // do something
}
```

Redundant Testing of Boolean Values

Problem: Computing BMI

Body Mass Index (BMI) is a measure of *health* on *weight*. It can be calculated by taking your *weight in kilograms* and *dividing* by *the square of your height in meters*. The interpretation of BMI for people 20 years or older is as follows:

BMI < 18.5: Underweight

18.5 ≤ BMI < 25.0: Normal

25.0 ≤ BMI < 30.0: Overweight

30.0 ≤ BMI: Obese

```
Enter your weight(kg):65
Enter your height(m):1.75
Your BMI is: 21.22 (Normal)
```

Problem: Computing BMI


```
import java.util.Scanner;
public class TestClass {
    public static void main(String[] args){
        // Enter your weight and height
        double bmi = weight / Math.pow(height, 2);
        String status = "Obese";
        if(bmi < 18.5){
            status = "Underweight";
        }else if(bmi < 25){
            status = "Normal";
        }else if(bmi < 30){
            status = "Overweight";
        }
        System.out.printf("Your BMI is: %4.2f (%s)",bmi,status);
    }
}
```

Logical Operators

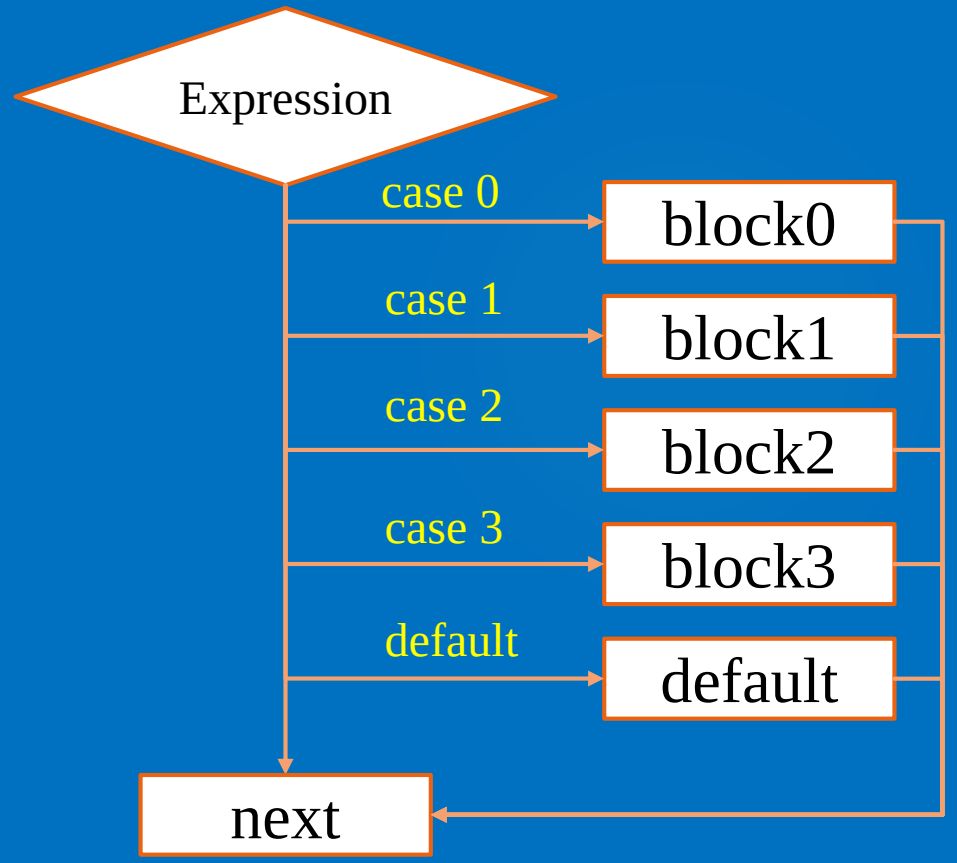
Operator	Name
!	not
&&	and
||	or
^	exclusive or

Case: A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

```
int year = 2020;
boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400
== 0);
```

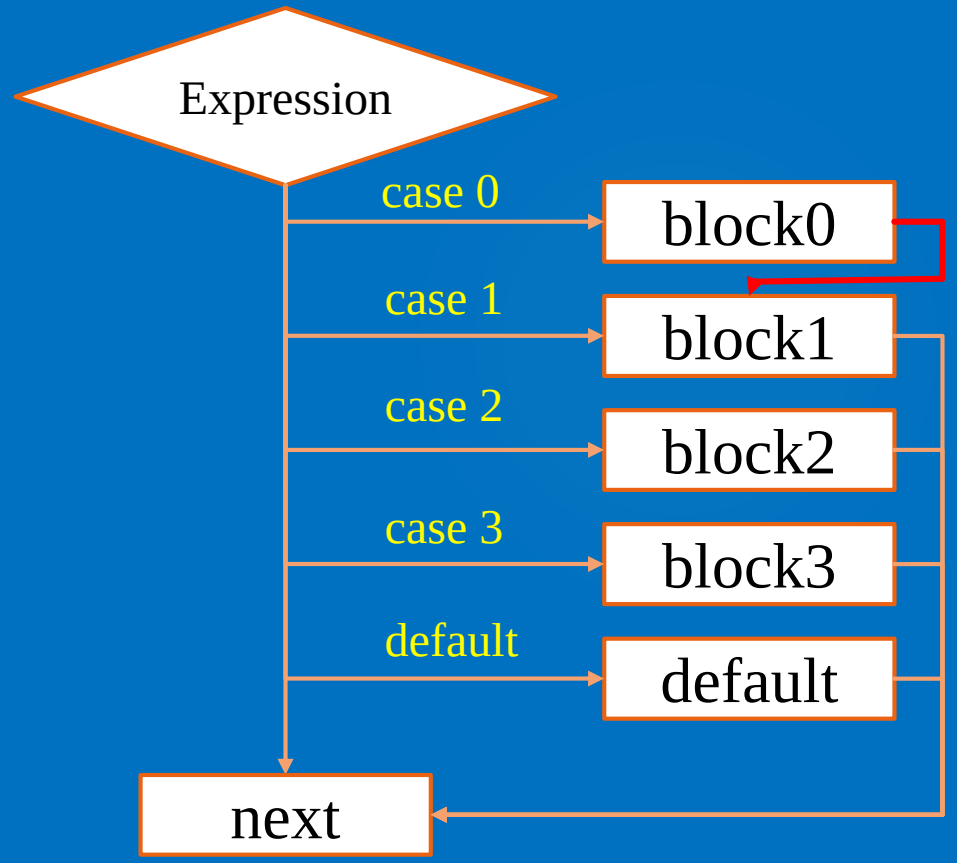
switch Statements


```
switch (expression) {
    case 0: block0;
    break;
    case 1: block1;
    break;
    case 2: block2;
    break;
    case 3: block3;
    break;
    default: default statements;
}
```



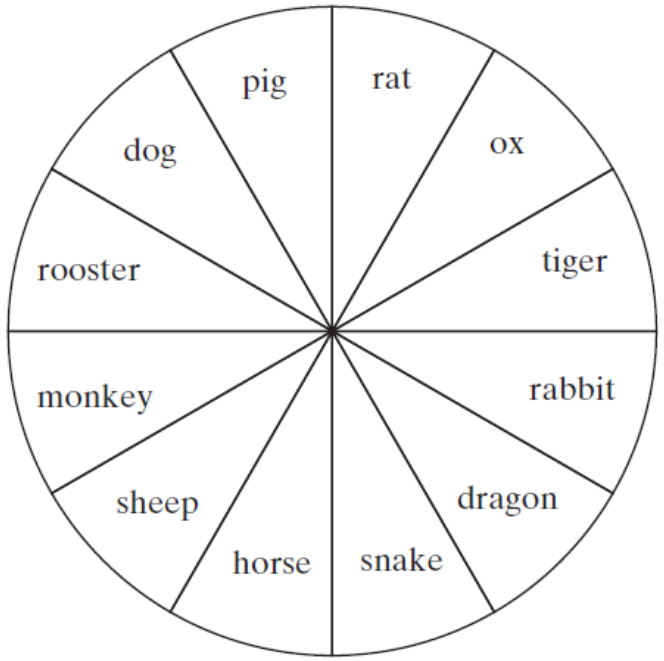
Missing break


```
switch (expression) {
    case 0: block0;
    //break;
    case 1: block1;
    break;
    case 2: block2;
    break;
    case 3: block3;
    break;
    default: default statements;
}
```



Problem: Chinese Zodiac

Now let us write a program to find out the Chinese Zodiac sign for a given year. The Chinese Zodiac is based on a ***twelve-year cycle***, with each year represented by an animal—monkey, rooster, dog, pig, rat, ox, tiger, rabbit, dragon, snake, horse, or sheep. Note that year % 12 determines the Zodiac sign. 1900 is the year of the rat because 1900 % 12 is 4.



year % 12 =

0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

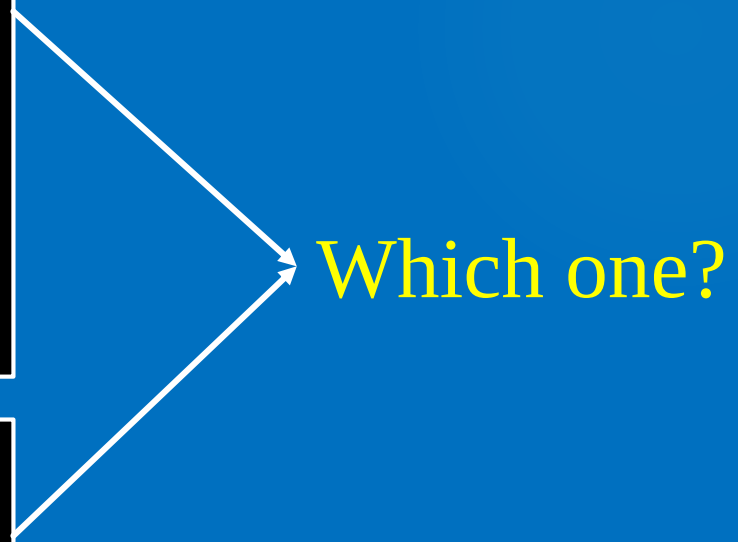
Problem: Chinese Zodiac


```
int year = input.nextInt();
switch (year % 12) {
    case 0: System.out.println("monkey"); break;
    case 1: System.out.println("rooster"); break;
    case 2: System.out.println("dog"); break;
    case 3: System.out.println("pig"); break;
    case 4: System.out.println("rat"); break;
    case 5: System.out.println("ox"); break;
    case 6: System.out.println("tiger"); break;
    case 7: System.out.println("rabbit"); break;
    case 8: System.out.println("dragon"); break;
    case 9: System.out.println("snake"); break;
    case 10: System.out.println("horse"); break;
    case 11: System.out.println("sheep");
}
```

Why use loops

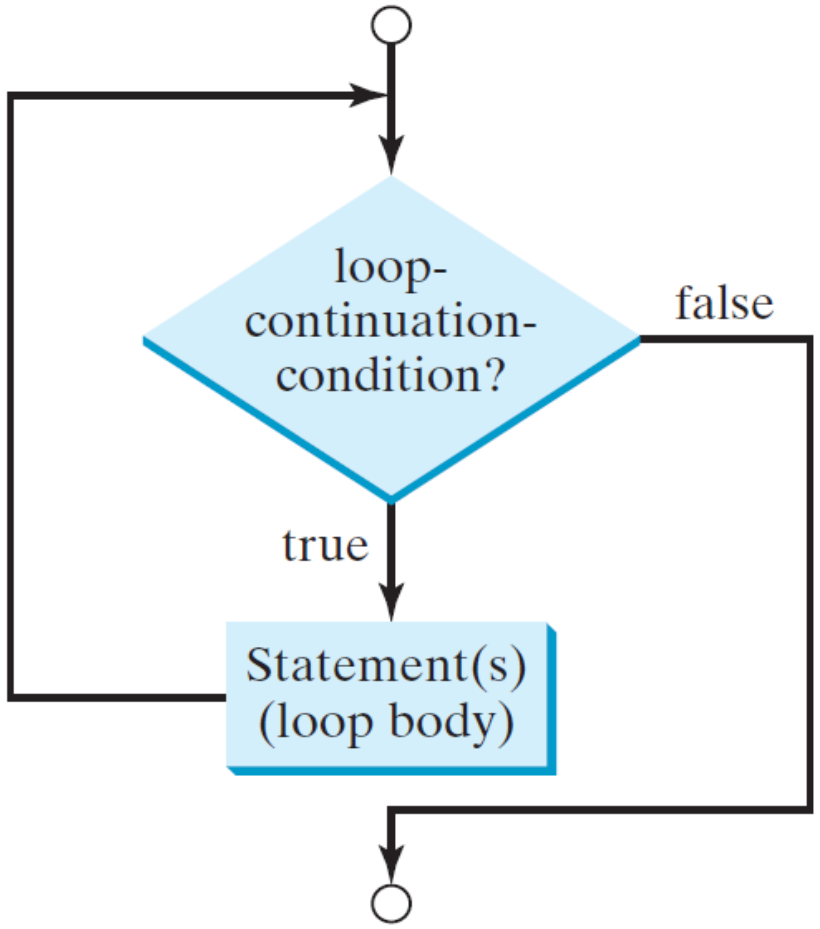

```
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
```

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

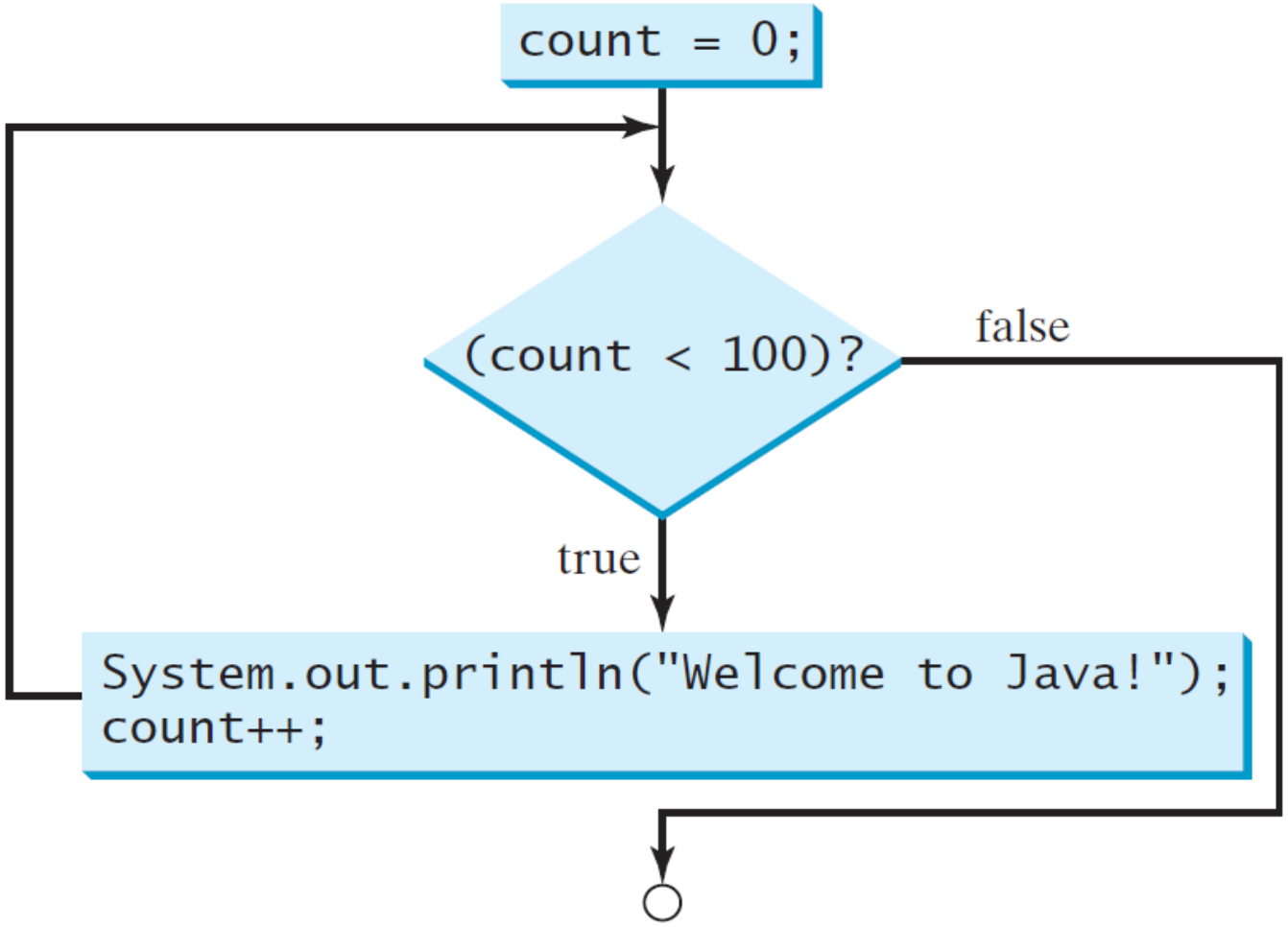


Which one?

while Loop Flow



(a)



(b)

while Loop Flow


```
while (loop-continuation-condition) {
    Statement(s);
}

int count = 0;
while (count < 10) {
    System.out.println("Hello World!");
    count++;
}
```

Problem: Guessing Numbers

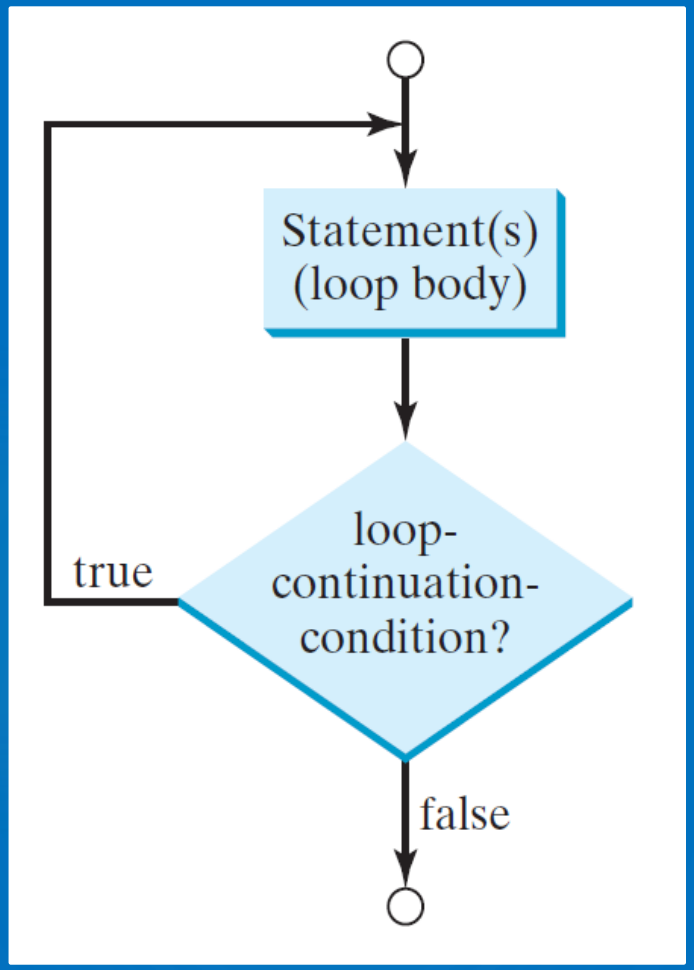
Write a program that *randomly generates* an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.

```
Guess a magic number between 0 and 100
Enter your guess: 50 ↵Enter
Your guess is too high
Enter your guess: 25 ↵Enter
Your guess is too low
Enter your guess: 42 ↵Enter
Your guess is too high
Enter your guess: 39 ↵Enter
Yes, the number is 39
```

Problem: Guessing Numbers


```
while (guess != number) {
    // Prompt the user to guess the number
    System.out.print("\nEnter your guess: ");
    guess = input.nextInt();
    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
} // End of loop
```

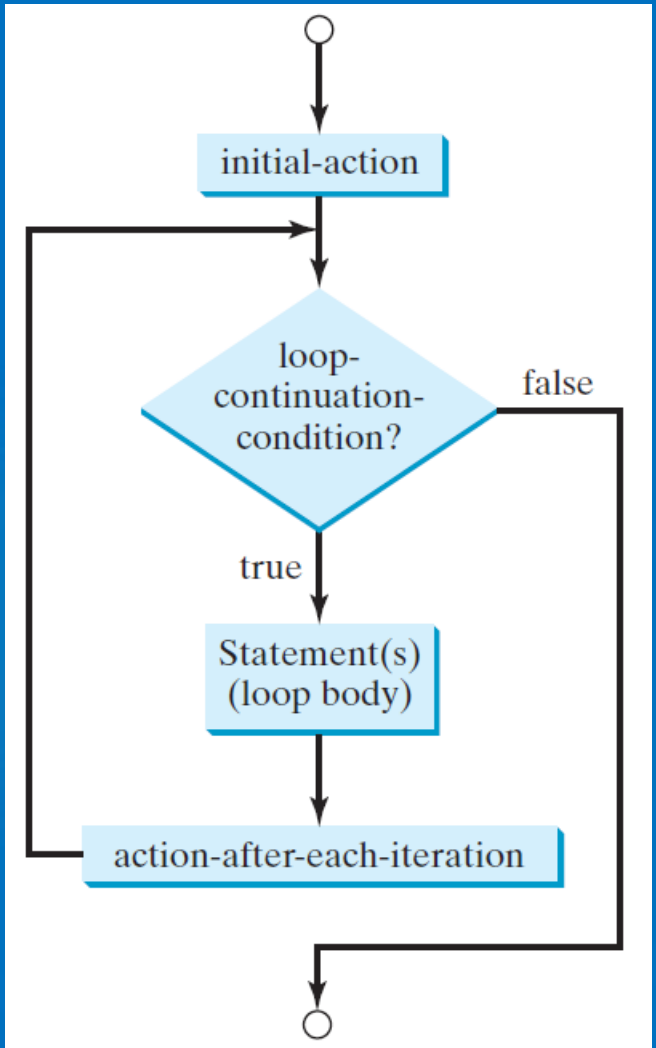
do-while Loop




```
do {
    // Loop body;
    Statement(s);
} while (expression);
```

```
int count = 0;
do{
    System.out.println("Hello World!");
    count++;
} while(count < 10);
```

for Loops




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

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

foreach syntax

Java SE5 introduces a new and more succinct for syntax, for use with *arrays* and *containers*. This is often called the *foreach syntax*, and it means that you don't have to create an int to count through a sequence of items—the foreach produces each item for you, automatically.

```
public static void main(String[] args) {
    int[] numList = {1, 2, 3, 4, 5};
    for(int x : numList) //for(type e: collection)
        System.out.println(x);
}
```

Nested Loops


```
public static void main(String[] args) {
    for(int i=1; i<=9; i++){
        System.out.print(i + " | ");
        for(int j = 1; j<=9; j++){
            System.out.printf("%-4d", i * j);
        }
        System.out.println();
    }
}
```

```
1 | 1   2   3   4   5   6   7   8   9
2 | 2   4   6   8   10  12  14  16  18
3 | 3   6   9   12  15  18  21  24  27
4 | 4   8   12  16  20  24  28  32  36
5 | 5   10  15  20  25  30  35  40  45
6 | 6   12  18  24  30  36  42  48  54
7 | 7   14  21  28  35  42  49  56  63
8 | 8   16  24  32  40  48  56  64  72
9 | 9   18  27  36  45  54  63  72  81
```

Note

The *initial-action* in a *for* loop can be *a list of* zero or more comma-separated *expressions*. The *action-after-each-iteration* in a *for* loop can be *a list of* zero or more comma-separated *statements*.

```
public class TestClass {
    public static void main(String[] args) {
        for(int i = 0, j = 0; i + j < 10; System.out.print(i++ + ++j)){

        }
    }
}
```

```
13579
```

Infinite Loops

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

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

(a)

Equivalent

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

(b)

Common Errors


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

while (i < 10); //wrong
{
    System.out.println("i is " + i++);
}

do {
    System.out.println("i is " + i++);
} while (i<10); //correct
```

Which Loop to Use?

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

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

(a)

Equivalent

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

(b)

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases:

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

(a)

Equivalent

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

(b)

Which Loop to Use?

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

break and *continue*

You can also control the flow of the loop inside the body of any of the iteration statements by using *break* and *continue*. *break quits the loop* without executing the rest of the statements in the loop. *continue stops* the execution of *the current iteration* and *goes back to the beginning* of the loop to begin the next iteration.

```
int num = 0;
while(true){
    num++;
    if(num > 10) break;
    if(num % 2 == 0) continue;
    System.out.print(num);
}
```

```
13579
```

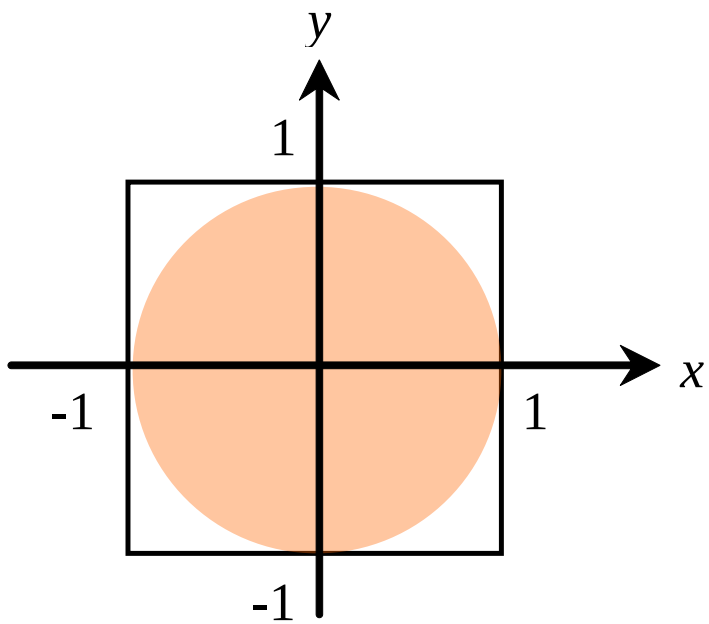
Problem: Predicating the Future Tuition

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

```
double initialTuition,tuition;
initialTuition = tuition = 10000;
int year = 1;
while(tuition < initialTuition * 2){
    tuition = tuition * 1.07;
    year++;
}
System.out.printf("After %d years, the tuition will be doubled", year);
```

Problem: Monte Carlo Simulation

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



circleArea / squareArea = π / 4

π = (circleArea / squareArea) * 4

circleArea / squareArea = N_{circle} / N_{square}

N_{circle}: The number of points in the circle

N_{square}: The number of points in the square

Problem: Monte Carlo Simulation


```
public static void main(String[] args){
    Scanner input = new Scanner(System.in);
    System.out.print("Enter sample number:");
    int totalPoints = input.nextInt();
    int pointsInCircle = 0;
    for(int i = 0; i < totalPoints; i++){
        double x = Math.random();
        double y = Math.random();
        if( Math.pow(x * x + y * y, 0.5) <= 1)
            pointsInCircle++;
    }
    double PI = 4 * (double)pointsInCircle / totalPoints;
    System.out.printf("PI is %5.3f\n", PI);
}
```