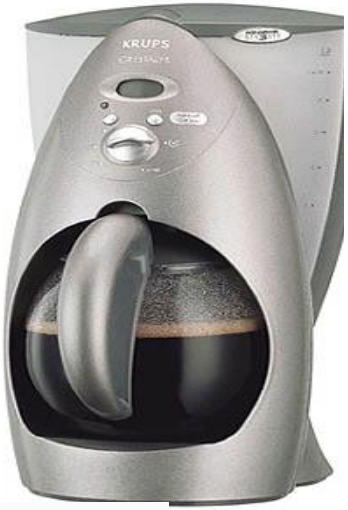
Lecture

Review Java

Introduction to Java

It was meant to!!

A programming language for appliances!

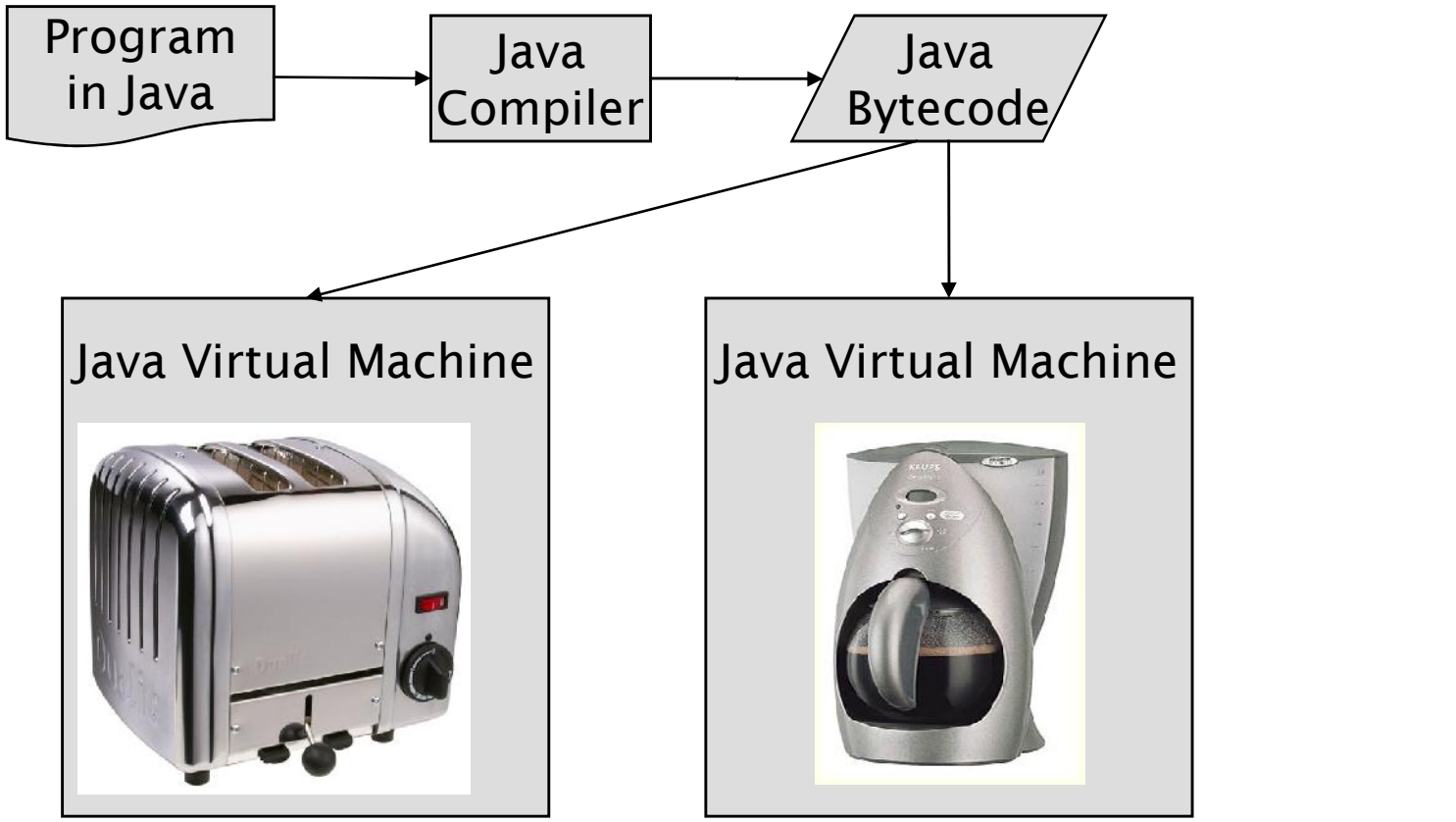


Must Run on Any Architecture

debug

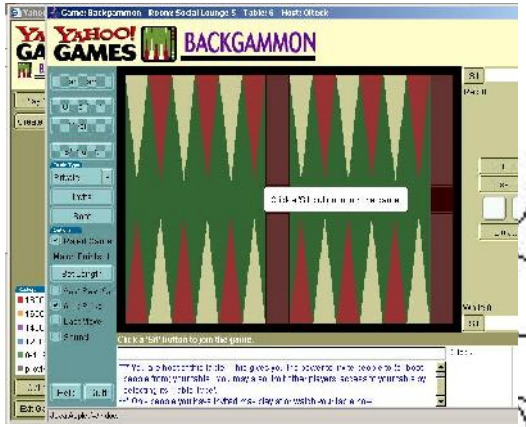
"WRITE ONCE, ~~RUN~~ ANYWHERE!"

pretty portable

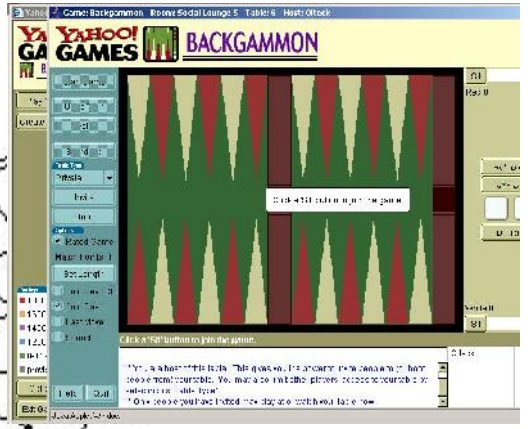


So What's Java Good For?

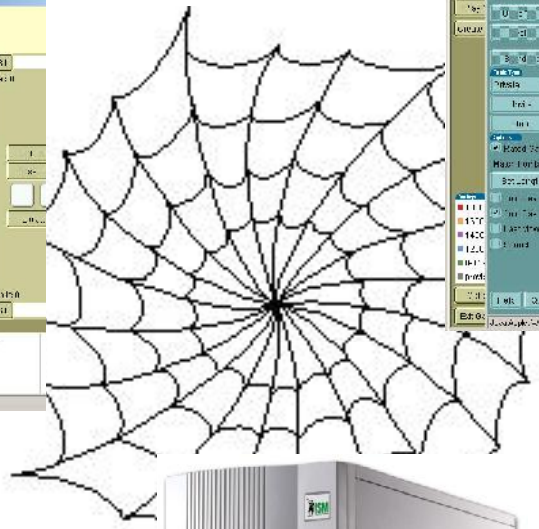
Web applications!



Java Applet



Java Applet



Server

The Java programming environment

- Compared to C++:
  - no header files, macros, pointers and references, unions, operator overloading, templates, etc.
- *Object-orientation*: Classes + Inheritance
- *Distributed*: RMI, Servlet, Distributed object programming.
- *Robust*: Strong typing + no pointer + garbage collection
- *Secure*: Type-safety + access control
- *Architecture neutral*: architecture neutral representation
- *Portable*
- *Interpreted*
  - *High performance through* Just in time compilation + runtime modification of code
- *Multi-threaded*

Java Features

- Well defined primitive data types: int, float, double, char, etc.
  - int 4 bytes [-2,147,648, 2,147,483,647]
- Control statements similar to C++: if-then-else, switch, while, for
- Interfaces
- Exceptions
- Concurrency
- Packages
- Name spaces
- Reflection
- Applet model

The Java programming environment

- Java programming language specification
  - Syntax of Java programs
  - Defines different constructs and their semantics
- *Java byte code*: Intermediate representation for Java programs
- *Java compiler*: Transform Java programs into Java byte code
- *Java interpreter*: Read programs written in Java byte code and execute them
- *Java virtual machine*: Runtime system that provides various services to running programs
- *Java programming environment*: Set of libraries that provide services such as GUI, data structures,etc.
- *Java enabled browsers*: Browsers that include a JVM + ability to load programs from remote hosts

Java: A tiny intro

- How are Java programs written?
- How are variables declared?
- How are expressions specified?
- How are control structures defined?
- How to define simple methods?
- What are classes and objects?
- What about exceptions?

How are Java programs written?

- Define a class HelloWorld and store it into a file: HelloWorld.java:

```
public class HelloWorld {
   public static void main (String[] args) {
      System.out.println("Hello, World");
   }
}
```

- **Compile HelloWorld.java**

```
javac HelloWorld.java
```

Output: HelloWorld.class

- **Run**

```
java HelloWorld
```

Output: Hello, World

Structure of Java programs

```
public class <name> {
    public static void main(String[] args) {
        <statement>;
        <statement>;
        ...
        <statement>;
    }
}
```

- Every executable Java program consists of a **class**
  - that contains a **method** named `main`
    - that contains the **statements** (commands) to be executed

Syntax

- **syntax**: The set of legal structures and commands that can be used in a particular programming language.

- some Java syntax:
  - every basic Java statement ends with a semicolon `;`
  - The contents of a class or method occur between `{` and `}`

System.out.println

- `System.out.println` : A statement to instruct the computer to print a line of output on the console.
  - pronounced "*print-linn*"
  - sometimes called a "*println statement*" for short

- Two ways to use `System.out.println` :

  ```
  System.out.println("<Message>");
  ```

  - Prints the given message as a line of text on the console.

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

  - Prints a blank line on the console.

How are variables declared?

Fibonacci:

```
class Fibonacci {
    public static void main(String[] arg) {
      int lo = 1;
      int hi = 1;
      System.out.println(lo);
      while (hi < 50) {
            System.out.println(hi);
            hi = lo + hi;
            lo = hi – lo;
      }
    }
}
```

Identifiers

- **identifier**: A name given to a piece of data, method, etc.
  - Identifiers allow us to refer to an item later in the program.
  - Identifiers give names to:
    - classes
    - methods
    - variables, constants (seen in Ch. 2)

- Conventions for naming in Java:
  - *classes*: capitalize each word (`ClassName`)
  - *methods*: capitalize each word after the first (`methodName`)
    (variable names follow the same convention)
  - *constants*: all caps, words separated by _ (`CONSTANT_NAME`)

Keywords

- **keyword**: An identifier that you cannot use because it already has a reserved meaning in the Java language.
- Complete list of Java keywords:

```
abstract    default     if          private      this
boolean     do          implements  protected    throw
break       double      import      public       throws
byte        else        instanceof  return       transient
case        extends     int         short        try
catch       final       interface   static       void
char        finally     long        strictfp     volatile
class       float       native      super        while
const       for         new         switch
continue    goto        package     synchronized
```

- You may not use `char` or `while` for the name of a class or method; Java reserves those to mean other things.
  - You could use `CHAR` or `While`, because Java is case-sensitive. However, this could be confusing and is not recommended.

How to define expressions?

- Arithmetic: +, -, *,/, %, =

  8 + 3 * 2 /4

  Use standard precedence and associativity rules

- Predicates: ==, !=, >, <, >=, <=

```
public class Demo {
    public static void main (String[] argv) {
        boolean b;
        b = (2 + 2 == 4);
        System.out.println(b);
    }
}
```

Declaring a static method

- Syntax for *declaring* a static method (writing down the recipe):

```
public class <class name> {

    public static void <method name>() {
        <statement>;
        <statement>;
        ...
        <statement>;
    }
}
```

- Example:

```
public static void printWarning() {
    System.out.println("This product is known to cause");
    System.out.println("cancer in lab rats and humans.");
}
```

How are control structures specified?

Typical flow of control statements: if-then-else, while, switch, do-while, and blocks

```
class ImprovedFibo {
  static final int MAX_INDEX = 10;
  public static void main (String[] args) {
    int lo = 1;
    int hi = 1;
    String mark = null;
    for (int i = 2; i < MAX_INDEX; i++) {
      if ((i % 2) == 0)
        mark =  " *";
      else mark = "";
      System.out.println(i+ ": " + hi + mark);
      hi = lo + hi;
      lo = hi - lo;
    }}}
```

Comments

- **comment**: A note written in the source code by the programmer to make the code easier to understand.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.

- Comment, general syntax:

  / <comment text; may span multiple lines> */*

  or,

  // <comment text, on one line>

- Examples:

```
/* A comment goes here. */
/* It can even span
   multiple lines. */
// This is a one-line comment.
```

Arrays

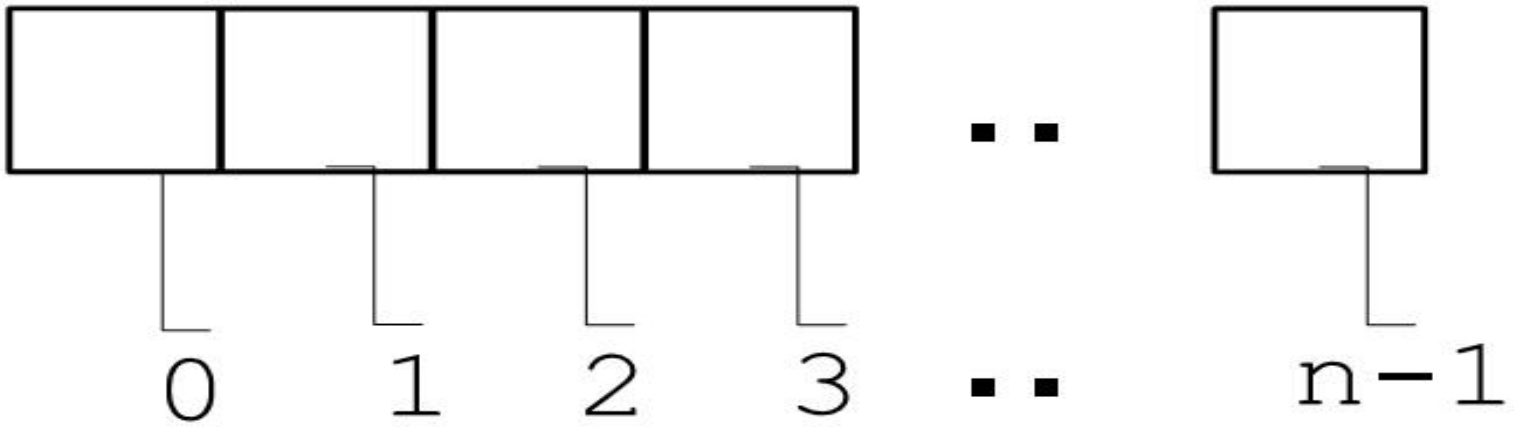
An array is an indexed list of values.

You can make an array of any type

int, double, String, etc..

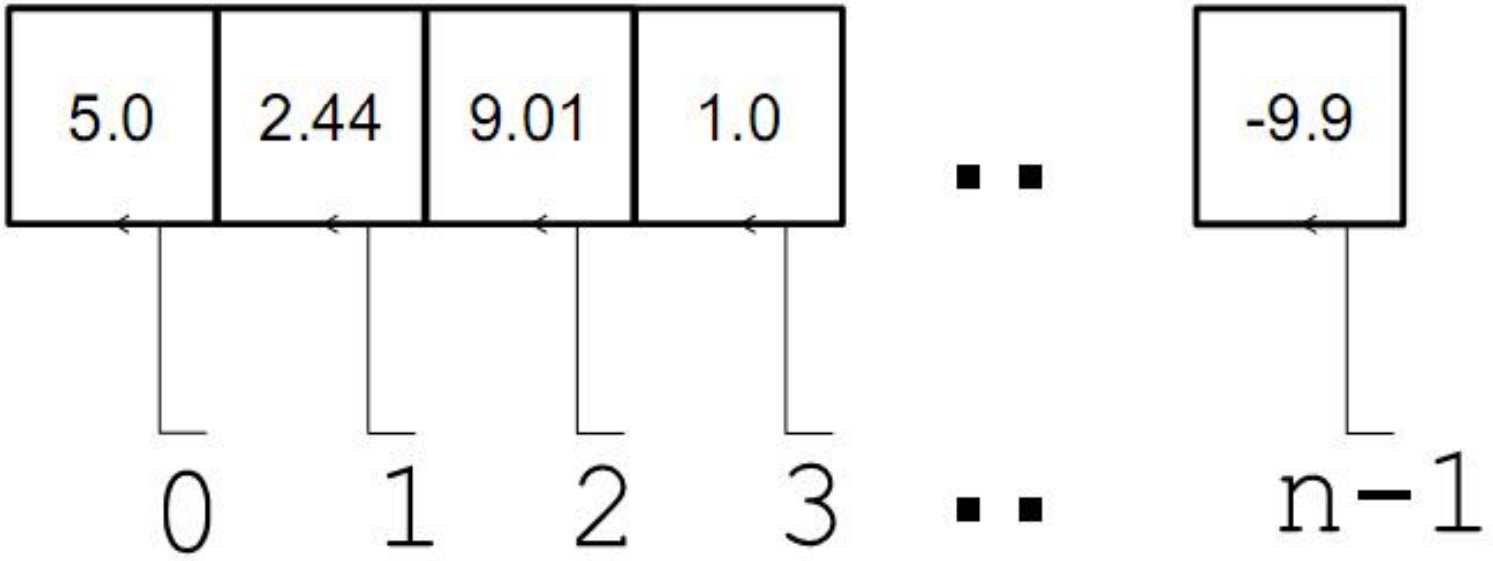
All elements of an array must have the same type.

Arrays



Arrays

Example: double [ ]



Arrays

The index starts at zero and ends at length-1.

Example:

```
int[] values = new int[5];
values[0] = 12; // CORRECT
values[4] = 12; // CORRECT
values[5] = 12; // WRONG!! compiles but
                // throws an Exception
                // at run-time
```

Have a demo with runtime exception

Arrays

An array is defined using TYPE [ ].

Arrays are just another type.

```
int[]   values;   // array of int

int[][] values;   // int[] is a type
```

Arrays

To create an array of a given size, use the operator `new` :

```
int[] values = new int[5];
```

or you may use a variable to specify the size:

```
int size = 12;
int[] values = new int[size];
```

Array Initialization

Curly braces can be used to initialize an array.

It can ONLY be used when you declare the variable.

```
int[] values  = { 12, 24, -23, 47 };
```

Accessing Arrays

To access the elements of an array, use the `[]` operator:

```
values[index]
```

Example:

```
int[] values  = { 12, 24, -23, 47 };
values[3] = 18;           // {12,24,-23,18}
int x = values[1] + 3; // {12,24,-23,18}
```

The *length* variable

Each array has a `length` variable built-in that contains the length of the array.

```
int[] values =  new int[12];
int size =      values.length; // 12

int[] values2 = {1,2,3,4,5}
int size2 =     values2.length; // 5
```

String arrays

A side note

```
public static void main (String[] arguments){
    System.out.println(arguments.length);
    System.out.println(arguments[0]);
    System.out.println(arguments[1]);
}
```

Looping through an array

Example 1:

```
int[] values = new int[5];

for (int i=0;  i<values.length; i++) {
   values[i] = i;
   int y = values[i] * values[i];
   System.out.println(y);
}
```

Looping through an array

Example 2:

```
int[] values = new int[5];
int i = 0;
while (i < values.length) {
   values[i] = i;
   int y = values[i] * values[i];
   System.out.println(y);
   i++;
}
```

End

Self Review

Additional Operators

Extended Assignment Operators

- The assignment operator can be combined with the arithmetic and concatenation operators to provide extended assignment operators. For example

```
int a = 17;
String s = “hi”;
a += 3;          // Equivalent to  a =  a + 3;
a -= 3;          // Equivalent to  a =  a – 3;
a *= 3;          // Equivalent to  a =  a * 3;
a /= 3;          // Equivalent to  a =  a / 3;
a %= 3;          // Equivalent to  a =  a % 3;
s += “ there”;   // Equivalent to  s =  s + “ there”;
```

Additional Operators

- Extended assignment operators can have the following format.

  variable op= expression;

  which is equivalent to

  variable = variable op expression;

- Note that there is no space between op and =.
- The extended assignment operators and the standard assignment have the same precedence.

Additional Operators

Increment and Decrement

- Java includes increment (++) and decrement (--) operators that increase or decrease a variables value by one:

```
int m = 7;
double x = 6.4;
m++;  // Equivalent to m = m + 1;
x--;    // Equivalent to x = x – 1.0;
```

- The precedence of the increment and decrement operators is the same as unary plus, unary minus, and cast.

Control Statements

- While and if-else are called ***control statemen***

```
while (some condition)
{
   do stuff;
}
```

Means do the stuff repeatedly as long as the condition holds true

Means if some condition is true, do stuff 1, and if it is false, do stuff 2.

```
if (some condition)
{
   do stuff 1;
}
else
{
   do stuff 2;
}
```

The if and if-else Statements

Principal Forms

- In Java, the if and if-else statements allow for the conditional execution of statements.

```
if (condition){
   statement;          //Execute these statements if the
   statement;          //condition is true.
}

if (condition){
   statement;          //Execute these statements if the
   statement;          //condition is true.
}else{
   statement;          //Execute these statements if the
   statement;          //condition is false.
}
```

The if **and** if-else **Statements**

- The indicated semicolons and braces are required
- Braces always occur in pairs
- There is no semicolon immediately following a closing brace.

The `if` and `if-else` Statements (cont.)

Additional forms:

```
if (condition)
    statement;

if (condition)
    statement;
else
    statement;
```

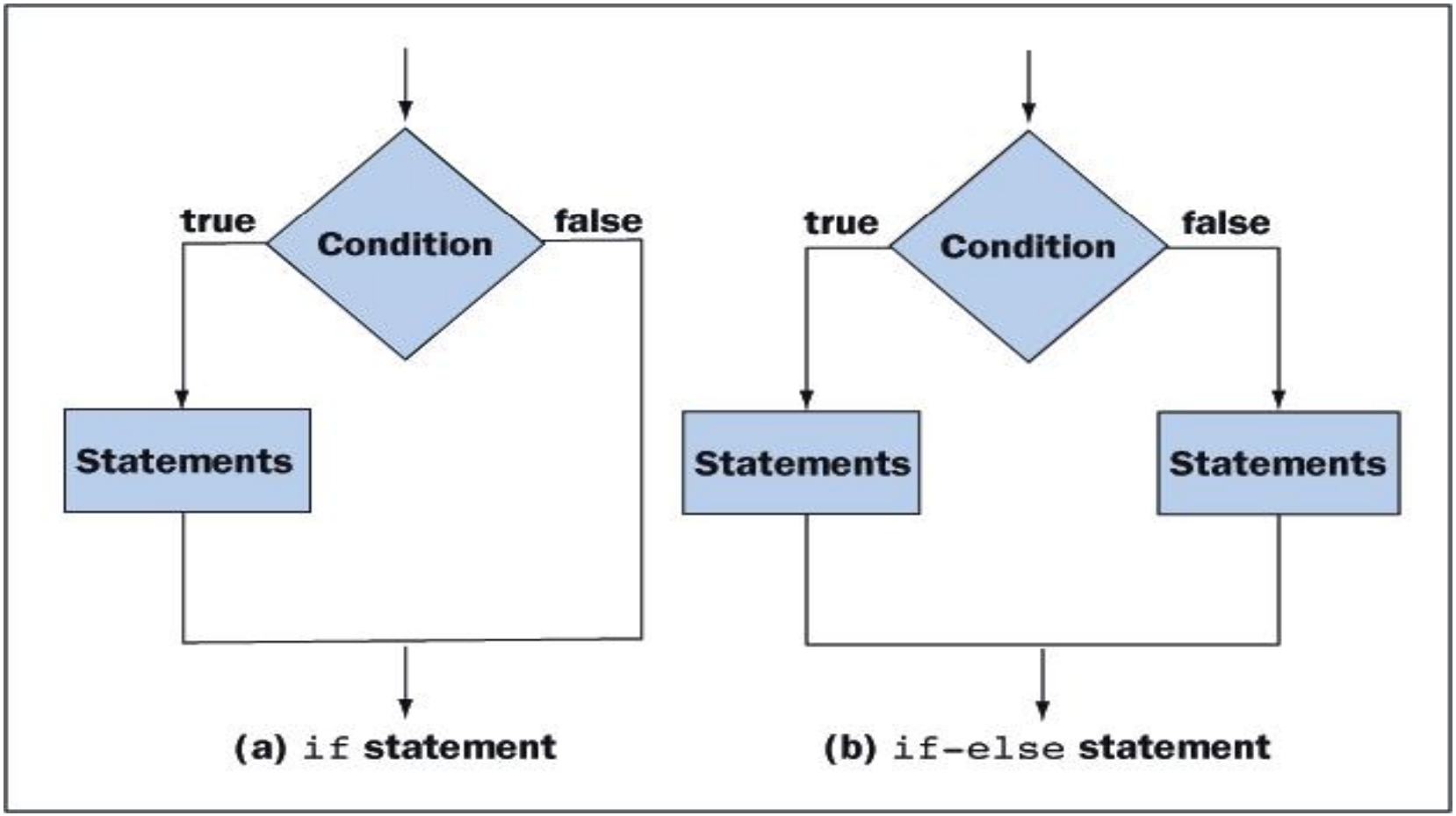
```
if (condition){
    statement;
        ...
    statement;
}else
    statement;

if (condition)
    statement;
else{
    statement;
        ...
    statement;
}
```

The `if` and `if-else` Statements (cont.)

- Better to over-use braces than under-use them
- Can help to eliminate logic errors
- Condition of an `if` statement must be a **Boolean expression**
- Evaluates to `true` or `false`
- A **flowchart** can be used to illustrate the behavior of `if-else` statements.

The if and if-else Statements

Figure 4-1 shows a diagram called a ***flowchart*** that illustrates the behavior of if and if-else statements.



The if **and** if-else **Statements**

Examples:

```
// Increase a salesman's commission by 10% if his sales are over $5000
if (sales > 5000)
   commission *= 1.1;

// Pay a worker $14.5 per hour plus time and a half for overtime
pay = hoursWorked * 14.5;
if (hoursWorked > 40){
   overtime = hoursWorked - 40;
   pay += overtime * 21.75;
}

// Let c equal the larger of a and b
if (a > b)
   c = a;
else
   c = b;
```

The if and if-else Statements

Relational Operators

Table 4-3 shows the complete list of relational operators available for use in Java.

OPERATOR	WHAT IT MEANS
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

The if **and** if-else **Statements**

The double equal signs (==) distinguish the equal to (equality comparison) operator from the assignment operator (=).

Side effects are caused if we confuse the two operators.

In the not equal to operator, the exclamation mark (!) is read as not.

Checking for Valid Input

Example 4.1: Computes the area of a circle if the radius >= 0 or otherwise displays an error message. Error trapping with if else statements work best in GUIs, for Terminal IO use while.

```
import java.util.Scanner;

public class CircleArea{

   public static void main(String[] args){
      Scanner reader = new Scanner(System.in);
      System.out.print("Enter the radius: ");
      double radius = reader.nextDouble();
      if (radius < 0)
         System.out.println("Error: Radius must be >= 0");
      else{
         double area = Math.PI * Math.pow(radius, 2);
         System.out.println("The area is " + area);
      }
   }
}
```

Avoid Sequential Selection

This is **not** a good programming practice.

Less efficient

only one of the conditions can be true

- these is called mutually exclusive conditions

```
if (condition1)       //avoid this structure
    action1           //use nested selection (see Ch. 6)
if (condition2)
    action2
if (condition3)
    action3
```

Compound Boolean Expressions

Logical Operators

And && (two ampersands) Conjunction

Or || (two pipe symbols) Disjunction

Not ! (one exclamation point) Negation

Use parentheses for each simple expression and the logical operator between the two parenthetical expressions.

i.e. ((grade >= 80) && (grade < 90))

Truth Tables for AND / OR

Expression1 (E1)	Expression2 (E2)	E1&&E2	E1||E2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Truth Table for NOT
Order of Priority for Booleans

Expression	Not E
(E)	!E
true	false
false	true

Order Of Priority in Boolean Expressions

1. !
2. &&
3. ||

Order of Operations including Booleans

1. ( )
2. !
3. *, /, %
4. +, -
5. <, <=, >, >=, = =, !=
6. &&
7. ||

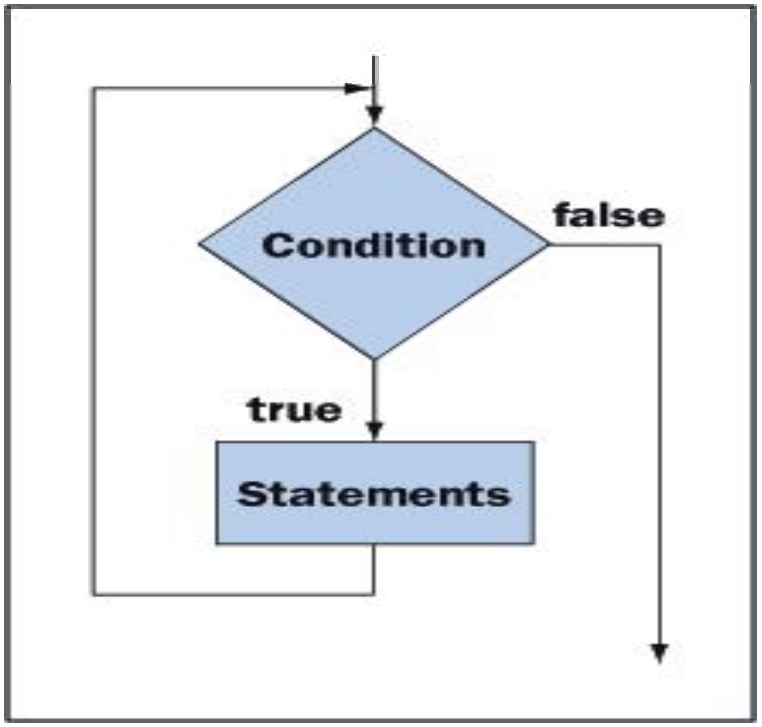
The while Statement

- The while statement provides a looping mechanism that executes statements repeatedly for as long as some condition remains true.

```
while (condition) //loop test
 statement;   //one statement inside the loop body

while (condition)        //loop test
{
statement;          //many statements
statement;          //inside the
...                 //loop body
}
```

The while Statement

- If the condition is false from the outset, the statement or statements inside the loop never execute. Figure 4-2 uses a flowchart to illustrate the behavior of a while statement.



The while Statement

Compute 1+2+…+100 (**Count-controlled Loops**)

- The following code computes and displays the sum of the integers between 1 and 100, inclusive:

```
// Compute 1 + 2 + … + 100
int sum = 0, counter = 1;  //Initialize sum and counter before loop
while (counter <= 100)
{
  sum += counter;  //point p (we refer to this location in Table 4-4)
   counter ++;          // point q (we refer to this location in table 4-4)
}
System.out.println  (sum);
```

The while Statement: count controlled loop

- The variable *counter* acts as a counter that controls how many times the loop executes.
- The *counter* starts at 1.
- Each time around the loop, it is compared to 100 and incremented by 1.
- The code inside the loop is executed exactly 100 times, and each time through the loop , sum is incremented by increasing values of *counter* .
- The variable *counter* is called the ***counter.***

The while Statement

Tracing the Variables

- To understand the lop fully, we must analyze the way in which the variables change on each pass or ***iteration*** through the loop. Table 4-4 helps in this endeavor. On the 100th iteration, cntr is increased to 101, so there is never a 101st iteration, and we are confident that the sum is computed correctly.

ITERATION NUMBER	VALUE OF CNTR AT POINT P	VALUE OF SUM AT POINT P	VALUE OF CNTR AT POINT Q
1	1	1	2
2	2	1 + 2	3
...	...	...	...
100	100	1 + 2 +...+ 100	101

The while Statement

Counting Backwards

- The counter can run backward.
- The next example displays the square roots of the numbers 25, 20, 15, and 10
- Here the counter variable is called number:

```
// display the square roots of 25, 20, 15, and 10
int number = 25;
while (number >= 10)
    {
System.out.println ("The square root of" + number
    + "is" + Math.sqrt (number));
number -= 5;
}
```

The while **Statement**

The output is:

```
The square root of 25 is 5.0
The square root of 20 is 4.47213595499958
The square root of 15 is 3.872983346207417
The square root of 10 is 3.162277660168379 5
```

The while Loop Accumulator

Write code that computes the sum of the numbers between 1 and 10.

```
int counter = 1;
int sum = 0;
while (counter <= 10)
{
   sum = sum + counter;
   counter = counter + 1;
}
```

The while Statement

Task-Controlled Loop

- Task-controlled loops are structured so that they continue to execute until some task is accomplished
- The following code finds the first integer for which the sum 1+2+…+n is over a million:

```
// Display the first value n for which 1+2+…+n
// Is greater than a million
int sum = 0;
int number = 0;
while (sum<= 1000000)
 {
   number++;
   sum += number;
}
system.out.println  (number);
```

The while Statement

Common Structure

- Loops typically adhere to the following structure:

```
initialize variables            //initialize (a "primed" while loop)
while (condition)
{                               //test
  perform calculations and      //loop
  change variables involved in the condition  //body
}
```

- In order for the loop to terminate, each iteration through the loop must move variables involved in the condition significantly closer to satisfying the condition.

Using while to Compute Factorials

- Example 4.2: Compute and display the factorial of *n*

```
import java.util.Scanner;

public class Factorial{

   public static void main(String[] args){
      Scanner reader = new Scanner(System.in);
      System.out.print("Enter a number greater than 0: ");
      int number = reader.nextInt();
      int product = 1;
      int count = 1;
      while (count <= number){
         product = product * count;
         System.out.println(product);
         count++;
      }
      System.out.println("The factorial of " + number +
                         " is " + product);
   }
}
```

The for Statement

- The for statement combines counter initialization, condition test, and update into a single expression.
- The form for the statement:

```
for  (initialize counter; test counter; update counter)
 statement;     // one statement inside the loop body

for (initialize counter; test counter; update counter)
{
 statement;     // many statements
 statement;     //inside the
 . . .;         //loop body
}
```

The for Statement

- When the for statement is executed, the counter is initialized.
- As long as the test yields true, the statements in the loop body are executed, and the counter is updated.
- The counter is updated at the bottom of the loop, after the statements in the body have been executed.

The for Statement

- The following are examples of for statements used in count-controlled loops:

```
// Compute 1 + 2 + … + 100

int sum = 0, counter;
for (counter = 1; counter <=100; counter ++)
 sum += counter;
System.out.println (sum);

// Display the square roots of 25, 20, 15, and 10
int number;
for (number = 25; number>=10; number-= 5)
 System.out.println (“The square root of”  + number +
        “is” + Math.sqrt (number));
```

The for Statement

Declaring the Loop Control Variable in a for Loop.

- The for loop allows the programmer to declare the loop control variable inside of the loop header.
- The following are equivalent loops that show these two alternatives:

```
int counter;            //Declare control variable above loop
for (counter = 1; counter <= 10; counter ++)
  System.out.println(counter);


for (int counter = 1; counter <= 10; counter ++)       //Declare
    control variable in loop header
  System.out.println(counter);
```

The for Statement

- Both loops are equivalent in function, however the second alternative is considered preferable on most occasions for two reasons:

1. The loop control variable is visible only within the body of the loop where it is intended to be used.
2. The same name can be declared again in other for loops in the same program.

for loop example

A for loop to calculate the sum of integers between a starting and ending value.

```
// Display the sum of the integers between a startingValue
// and an endingValue, using a designated increment.

int cntr, sum, startingValue, endingValue, increment;

startingValue = 10;
endingValue = 100;
increment = 7;

sum = 0;
for (cntr = startingValue; cntr <= endingValue; cntr += increment)
   sum += cntr;
System.out.println (sum);
```