Learning to move from one language to another is a fact of life for today’s software professionals. Whether or not C++ is your first programming language, it will undoubtedly not be your last. It is not uncommon for a software professional to be fluent in at least a half dozen or more programming languages of various types. Categories of languages include general purpose languages (C++, Java, C#, Smalltalk), scripting languages (Perl, Python), Web-based languages (Javascript, PHP, Curl), functional languages (ML, Haskell), AI languages (Common Lisp, Prolog), and many others.

Java and C++ are currently the two most commonly used general purpose programming languages. For this reason many programmers find it useful to have at least a passing familiarity with both languages. Fortunately, Java was inspired by C++ and many features, such as the control flow statements if, while, and for, were adopted almost without change.

In this appendix we will describe the most basic differences between the two languages. Further information on the Java language can be found in [1]. Information on moving to C++ if Java is your first programming language can be found in [2].

Data Types and Variables

In C++ the number of bits used by an integer is implementation dependent. While many compilers use 32-bit integers, there are some machines that use 16-, 20-, 36-,
APPENDIX H  ■  A C++/Java Comparison

or even 64-bit integers. In Java the language specifies that an integer must be exactly 32 bits.

The Boolean type is called boolean in Java, not bool as in C++.

The Java string type is called String. It is similar to the C++ string type, but note the following differences.

- Java strings cannot be modified (they are implicitly const, in C++ terms).
- Java strings store 16-bit Unicode characters, not 8-bit ASCII characters.
- You can concatenate strings with any type of object in Java, while in C++ strings can only be combined with strings.
- To compare strings in Java, use the member functions equals or compareTo. Both take an argument string. The function equals returns a Boolean, while compareTo returns an integer value that is less than zero if the first string is smaller than the second, zero if they are equal, and greater than zero if the first is larger than the second.

```java
String p = "abc";
String q = "pqr";
if (p.compareTo(q) < 0) . . .
```

- Many member functions in the Java string class have functionality (substrings, indexing) similar to, but names different from their C++ counterparts.

Java does not have explicit pointer values or references, although internally pointers are used extensively (see the section on Objects, below).

Variables, Constants, and Functions

The Java compiler is obligated to check that local variables and data fields are initialized before they are used. This eliminates a common error in C++ programs.

It is not possible to create global variables or functions in Java that are not associated with a class. That is, the only functions in Java are member functions. Member functions in Java are often called methods.

Java does not have the const reserved word. An analogous reserved word, final, means that a value can not be reassigned. While not exactly the same (a final value can change its internal state, while a const in C++ cannot change in any way), the two are frequently used in the same fashion.

Classes

Although classes in C++ and Java are similar in intent, there are a number of superficial differences between them:
APPENDIX H ▪ A C++/Java Comparison

- In Java inheritance is signaled by the reserved word `extends`, rather than a colon.
- In Java the visibility modifiers `public`, `protected`, and `private` are attached to each data field and member function independently, rather than dividing the class definition into sections.
- All classes inherit from a base class in Java. If no explicit base class is specified, the class `Object` is used as a base class. Therefore, all classes ultimately inherit from a single base class named `Object`.
- The member function bodies are placed directly in the class definition, rather than being written separately.
- There is no semicolon at the end of a class.
- There is no `virtual` reserved word. All member functions are implicitly `virtual` (and thereby potentially subject to being overridden).
- There is no field initializer list. Data fields are initialized by an assignment statement in the body of a constructor. To pass data values to the constructor for a base class, the reserved word `super` is used.
- The reserved word `super` is also used to refer to a base class when a derived class wishes to invoke an overridden member function. This replaces the use of qualified names in C++ programs. The member function `changeSalary` in the example below illustrates this use.

The following is an example Java class definition:

```java
public class Manager extends Employee
{
    public Manager(String name, int salary, double bp)
    {
        super(name, salary); // Invoke constructor for Employee base class
        bonus_percent = bp;
        bonus = bp * salary;
    }

    void changeBonus(double new_bp)
    {
        bonus_percent = new_bp;
        bonus = bonus_percent * salary;
    }

    void changeSalary(int new_salary)
    {
        super.changeSalary(new_salary); // Invoke function from base class
        bonus = bonus_percent * new_salary;
    }

    private int bonus;
    private double bonus_percent;
}
```
4  APPENDIX H ▪ A C++/Java Comparison

**Interfaces**

Java supports a concept called an *interface*. An interface looks superficially like a class, but uses the reserved word *interface* rather than *class*. An interface cannot include variable data fields (although it can include constants—that is, data fields declared using the modifier *final*), and does not provide implementations for member functions. In C++ terms, it is as if each member function were a pure virtual. An interface is therefore a description of a set of desired behaviors, with no implementation. The following illustrates the syntax:

```java
/**
   Interface for collections that can be searched.
*/
interface LookUp
{
    /**
       @param name  the name being examined
       @return true if the name is part of the collection
    */
    public boolean contains(String name);

    /**
       @param name  the name of the value desired
       @return object associated with a given name
    */
    public Object find(String name);
}
```

A class indicates that it supports the interface with the *implements* reserved word. It must then provide an implementation of each method defined by the interface.

```java
class SearchableContainer implements LookUp
{
    public boolean contains(String name)
    {
        /* Implementation here */
    }
}
```

Java does not support multiple inheritance of classes. However, it does allow a class to implement multiple interfaces.

**Objects**

The most notable difference between Java and C++ is the treatment of objects. In Java object variables are treated internally as pointers. The Java literature refers to object variables as references, although they are not exactly the same as C++ references. (For example, Java object variables can be reassigned, whereas C++ references cannot. Java object variables can be null, but C++ references can never be
A C++/Java Comparison

NULL.) This means that all Java objects must be created using the new reserved word, as in the following:

```java
Employee sarah = new Employee("Sarah Smith", 67000);
```

Parentheses are required even if there are no arguments being passed to the constructor. Note carefully that the variable is declared as a simple name, not as a pointer. Java does not have the pointer/non-pointer distinction. Even arrays must be created using the new operator. The size is not included as part of the declaration, but instead as part of the new operation:

```java
Employee[] department = new Employee[10]; // Create 10 new Employee values
```

The number of elements in an array can be determined using a data field named length, as in department.length. Assignment in Java is equivalent to the assignment of pointers in C++. That is, assignment establishes the two values as referring to the same object, instead of making a true copy as in C++:

```java
Manager fred = new Manager("Fred Smith", 47000, 120);
Manager james = fred; // Now they refer to the same object.
james.changeBonus(200); // Now fred’s bonus is also changed.
```

To make a true copy in Java you must explicitly define a member function that returns a clone.

**Functions**

In Java every function is associated with a class. There are no global variables, nor ordinary (that is, nonmember) functions. To execute a Java program you specify the starting class, and the function named main in that class is invoked. The function main does not return a value in Java.

Function invocations pass object references by value. There is no equivalent to the C++ pass by value or pass by reference. This is illustrated by the following:

```java
class RewardSystem {
   // Functions can only be found in classes
   public void yearEndUpdate(Manager man)
   {
      man.changeBonus(0.35); // Update bonus to 35%
      man = new Manager("Sally Jones", 47000, 23);
      // Unlike C++ pass by reference, will NOT change the value of the
      // actual argument.
   }
}
RewardSystem man_class = new RewardSystem();
Manager fred = new Manager("Fred Smith", 47000, 120);
man_class.yearEndUpdate(fred); // Will have the effect of changing
   // the value fred.bonus
```
Arrays and Array Lists

All Java arrays are allocated on the heap:

```java
String[] names = new String[100];
```

Note that the [] is placed with the type, not the variable name.

The analog to the C++ vector is the ArrayList. You use the get and set methods, not the [] operator, to access the elements:

```java
ArrayList<String> names;
names.add("Harry"); // The analog of push_back
names.add("Lisa");
for (int i = 0; i < names.size(); i++)
  {
    String name = names.get(i);
    names.set(i, "***");
  }
```

Memory Management

In C++ the programmer is explicitly required to manage dynamic memory using the new and delete operators. Java includes a garbage collection system as part of its run-time library. The garbage collector monitors a running program and automatically recovers memory when it can no longer be used. This eliminates many memory leaks that can plague C++ programs.

Exception Handling

Exception handling is much more tightly integrated in Java than it is in C++. There are two categories of exceptions: checked exceptions (similar to runtime_error in C++), and unchecked exceptions (similar to logic_error in C++). Member functions that potentially can throw a checked exception must declare this using the throws reserved word. The compiler ensures that an invocation of such a function occurs in the context of a try/catch statement that will handle the exception. Values thrown must be objects from a class that is derived from a system class named Throwable. There are other small differences in the various forms the catch statement; these are described in [2].

```java
public class Employee
{
  . . .
  void initializeFromFile(File f) throws IOException
  {
    if (. . .)
      throw new IOException("Cannot Initialize");
  }
```
```java
Employee fred = new Employee();
File fin = new File("fred.data");
try {
    fred.initializeFromFile(fin);
} catch (IOException e) {
    System.out.println("Received I/O Exception " + e);
} catch (Exception e) // Catch any other type of exception
    {
    System.out.println("Received other exception " + e);
}
```

**Standard Library**

While the Java language is arguably smaller and simpler than C++, the same cannot be said for the standard library. The Java standard library is enormous, covering such features as internationalization, networking, mathematics, sound, Web applications and services, databases, and much more. Because the library is continually being added to, no reference can be entirely comprehensive. Most programmers are familiar with only a tiny fraction of this library. The most up-to-date information can be found at Sun’s official Java Web site, [http://java.sun.com](http://java.sun.com).

**Further Reading**