Part 5 Programming Languages

Text 1: About Programming Languages

Just as there are many different languages in use throughout the world for humans to communicate with, there are different languages that are used to communicate with a computer. Programming languages, in computer science, are the artificial languages used to write a sequence of instructions (a computer program) that can be run by a computer. They are standardized communication technique for expressing instructions to a computer. Similar to natural languages, such as English, programming languages have a vocabulary, grammar, and syntax. However, natural languages are not suited for programming computers because they are ambiguous, meaning that their vocabulary and grammatical structure may be interpreted in multiple ways. The languages used to program computers must have simple logical structures, and the rules for their grammar, spelling, and punctuation must be precise. They enable a programmer to precisely specify what data a computer will act upon, how the data will be stored or transmitted and precisely what actions to take under various circumstances.

1. The Development of Programming Languages

Programming languages have been under development for years and will remain so for many years to come. Perhaps the languages of tomorrow will be more natural with the invention of quantum and biological computers. The following are detailed descriptions for five generation languages.

2. First Generation: Machine Languages

It has been shown that a program instruction comprises a particular combination of binary digits. Different makes and types of computers use different "codes" of binary digits to represent instructions. Such codes are referred to as machine or instruction codes. At this level, a computer has a basic repertoire of instructions that it can perform, known as the instruction set. Typically this instruction set includes:

- Basic arithmetic operations,
- Comparators of various kinds (e.g. equality operators and so on),
- Facilities to deal with sequences of characters,
- Input/output operators.

In the early days of computer programming, all programs had to be written in machine code. For example, a short (3 instruction) program might look like this:

0111 0001 0000 1111
1001 1101 1011 0001

1110 0001 0011 1110

An instruction in machine language generally tells the computer four things: (1) where to find one or two numbers or simple pieces of data in the main computer memory (Random Access Memory, or RAM), (2) a simple operation to perform, such as adding the two numbers together, (3) where in the main memory to put the result of this simple operation and (4) where to find the next instruction to perform. While all executable programs are eventually read by the computer in machine language, they are not all programmed in machine language. It is extremely difficult to program directly in machine language because the instructions are sequences of 1s and 0s.

Machine code has several significant disadvantages associated with it:

- It is not intuitively obvious what a machine code instruction does simply from its encoding. Consequently it is very difficult to read and write machine code.
- The writing of machine code is extremely time consuming and error prone.
- Many different machine codes exist (one for each make and type of computer).

These disadvantages all serve to severely limit the applications to which computers can be applied when using machine code.

3．Second Generation: Assembler Languages

Assembler languages were initially developed to address the disadvantages associated with machine code programming. They used symbolic codes instead of lists of binary instructions. Consequently programming became more “friendly”. An example of assembly code is given below:

MOV AX 01
MOV BX 02
ADD AX BX

In assembler language each line of the program corresponds to one instruction in machine code. For a program written in an assembler language to be executable, it must be translated into machine code using a translating program called an assembler.

Assembly languages share certain features with machine languages. For instance, it is possible to manipulate specific bits in both assembly and machine languages. Programmers use assembly languages when it is important to minimize the time it takes to run a program, because the translation from assembly language to machine language is relatively simple. Assembly languages are also used when some part of the computer has to be controlled directly, such as individual dots on a monitor or the flow of individual characters to a printer.

Although use of assembly languages offers some advantages, there are still a number of significant disadvantages associated with their use: (1) each model of computer has its own assembly language associated with it, (2) assembler programming still requires great attention to detail and hence remains both time consuming and tedious, and (3) the risk of program error is not significantly reduced.

Note that there are some computer applications, such as interfacing with peripherals, where an assembler language is still a necessity.

4. Third Generation: High Level Languages

From the early 1950s onwards high-level languages were developed with the express aim of providing the means whereby computer programs could be written more efficiently and in a less error prone manner. High-level languages are relatively sophisticated sets of statements utilizing words and syntax from human language. They are more similar to normal human language than assembly or machine languages and are therefore easier to use for writing complicated programs. These programming languages allow larger and more complicated programs to be developed faster. The advantages offered are as follows.

- Programs written in a high-level language are more adapted to human modes of expression than to the computer's set of instructions. Programs are expressed in "half-English", and arithmetic calculations are written in a way familiar to mathematics.
- Programmers can concentrate more closely on the problem to be solved rather than the mass of detail required for machine code or assembly language programming.
- High-level languages are not necessarily dedicated to a particular type or model of computer, a feature known as portability.

Ultimately, a computer can only operate on programs defined using machine code. Consequently, a program written in a high-level language such as Java cannot be run directly. To execute a computer program written in a high level language it must be either compiled or interpreted. For this reason, programs written in a high-level language may take longer to execute and use up more memory than programs written in an assembly language.

1) Compiler

A compiler is software that looks at an entire high-level program before translating it into machine language. The programming instructions of a high-level language are called the source code. The central task of a compiler is to translate source code written in a high level language into a machine form, which in this case is called the object code. The advantage is that the machine executable form runs much faster than if it were interpreted (see below). The disadvantage is that different machines and operating systems have different machine codes associated with them. Generally, compilers have two functions:

- Checking for syntactic errors of source code
- Translating source code into object code

Examples of high-level languages using compilers are C and C++.

2) Interpreter

In the case of interpretation, each line of the program is decoded and "interpreted" by a special program known as an interpreter. Different interpreters are required for different languages (and different machines). Interpretation occurs every time a line in a program is executed. This means that a line which occurs many times must be interpreted on each occasion. This wastes computer time, and causes programs to run relatively slowly.

However, the repeated examination of the source program by an interpreter allows interpretation to be more flexible than when using a compiler. Interpreters also provide a faster

and easier way of testing small programs or fragments of programs. Further, in the context of error detection, because the interpreter works directly with the source code, errors can be reported accurately with reference to line numbers (this is not the case when programs are compiled). Another, more questionable advantage, is that parts of a program which are not executed need not be interpreted.

5．Fourth Generation: Very High Level Languages

Compared with third generation languages, 4GLs, for forth generation languages, are much more user-oriented and allow programmers to develop programs with fewer commands, although they require more computing power. 4GLs are called nonprocedural because a programmer and even users can write programs that only tell the computer what they want done, without specifying all the procedures for doing it. 4GLs consist of report generators, query languages, application generators, and interactive database management system languages.

- Report generator also called a report writer, is a program for end-users that are used to produce a report. Report generators were the precursor to today's query language.
- Query language is an easy-to-use language for retrieving data from a database management system.
- Application generator is a programmer's tool that generates applications programs from descriptions of the problem rather than by traditional programming. The benefit is that the programmer does not need to specify how the data should be processed.

4GLs may not entirely replace third generation languages because they are usually focused on specific tasks and therefore offer fewer options. Still, they improve productivity because programs are easy to write.

6．Fifth Generation: Artificial Intelligence

Fifth-generation language is artificial intelligence, making computers behave like humans. The term was coined in 1956 by John McCarthy at the Massachusetts Institute of Technology.

New Words and Expressions

ambiguous　*adj.* 含糊不清的，模棱两可的
punctuation　*n.* 标点；标点法；标点符号的使用
statement　*n.* 语句
precursor　*n.* 先驱者，前导，先进者
debug　*vt.* 驱除（某处的）害虫；排除程序等中的错误
portable　*adj.* 可移植的；便携的
quantum　*n.* 量，额；定量，定额；份；总量
assembly language　汇编语言

Exercises to the Text

1．Translate the following words and phrases into English.

（1）程序设计语言　（2）计算机科学　（3）人工语言　（4）自然语言　（5）电子设备

（6）面向对象编程 （7）机器语言 （8）汇编语言 （9）高级语言 （10）目标代码

2．Translate the following paragraphs into Chinese.

(1) Similar to natural languages, such as English, programming languages have a vocabulary, grammar, and syntax. However, natural languages are not suited for programming computers because they are ambiguous, meaning that their vocabulary and grammatical structure may be interpreted in multiple ways.

(2) The languages used to program computers must have simple logical structures, and the rules for their grammar, spelling, and punctuation must be precise. They enable a programmer to precisely specify what data a computer will act upon, how the data will be stored or transmitted and precisely what actions to take under various circumstances.

(3) An instruction in machine language generally tells the computer four things: ①where to find one or tow numbers or simple pieces of data in the main computer memory (Random Access Memory, or RAM), ②a simple operation to perform, such as adding the two numbers together, ③where in the main memory to put the result of this simple operation and ④where to find the next instruction to perform.

(4) Assembly languages share certain features with machine languages. For instance, it is possible to manipulate specific bits in both assembly and machine languages. Programmers use assembly languages when it is important to minimize the time it takes to run a program, because the translation from assembly language to machine language is relatively simple. Assembly languages are also used when some part of the computer has to be controlled directly, such as individual dots on a monitor or the flow of individual characters to a printer.

(5) High-level languages are relatively sophisticated sets of statements utilizing words and syntax from human language. They are more similar to normal human language than assembly or machine languages and are therefore easier to use for writing complicated programs. These programming languages allow larger and more complicated programs to be developed faster.

Text 2: C

C was developed in the early 1970s, and it was developed by Dennis Ritchie as a systems programming language for UNIX. C has grown into a very popular language now. C might best be described as a “medium level language”. Like a true high level language, there is a one-to-many relationship between a C statement and the machine language instructions it is compiled into. However, unlike most high level languages, C let you easily do chores (such as bit and pointer manipulation) additionally performed by assembly languages. Therefore, C is an especially good tool to use for developing operating systems (such as the UNIX operating system), or other system software.

1．What is C

The C programming language is a popular and widely used programming language for creating computer programs. It is one of thousands of programming languages currently in use. C has been around for several decades and has won widespread acceptance because it gives

programmers maximum control and efficiency. If you are programmer, or if you are interested in becoming a programmer, there are a couple of benefits you gain from learning C:

- You will be able to read and write code for a large number of platforms—everything from microcontrollers to the most advanced scientific systems can be written in C, and many modern operating systems are written in C.
- The jump to the object oriented C++ language becomes much easier. C++ is an extension of C, and it is nearly impossible to learn C++ without learning C first.

Lexically, C is more cryptic than other languages. In fact, C is an easy language to learn. It is a bit more cryptic in its style than some other languages, but you get beyond that fairly quickly. For example, brackets are often used to obviate the need for keywords. However, underlined characters are allowed in identifiers, which can make them more understandable.

There are a number of monadic and addict (called binary) operators. Some have unexpected precedence. Brackets may be ignored by the compiler, with occasionally surprising results. There are shift operations. Overflows on integer arithmetic may be ignored. There are some composite symbols with special meanings: for example "&&" means "and then" and "||" means "or else".

There are several integer types of different sizes and there are floating point numbers, * pointers (C talks of indirection), arrays and structures. C is not strongly typed: for example, some compilers do not insert run-time checks on array subscripts, etc. Type conversion is permissive. Address arithmetic can be performed on pointers; Null is demoted by a zero value.

C has procedures and functions. There are few features (apart from procedures and functions) to support modularization, however; separate (strictly independent) compilation is allowed.

C is what is called a compiled language. This means that once you write your C program, you must run it through a C compiler to turn your program into an executable that the computer can run (execute). The C program is the human-readable form, while the executable that comes out of the compiler is the machine-readable and executable form. What this means is that to write and run a C program, you must have access to a C compiler. If you are using a UNIX machine (for example, if you are writing CGI scripts in C on your host's UNIX computer, or if you are a student working on a lab's UNIX machine), the C compiler is available for free. It is called either "cc" or "gcc" and is available on the command line. If you are a student, then the school will likely provide you with a compiler—find out what the school is using and learn about it. If you are working at home on a Windows machine, you are going to need to download a free C compiler or purchase a commercial compiler. A widely used commercial compiler is Microsoft's Visual C++ environment (it compiles both C and C++ programs). Unfortunately, this program costs several hundred dollars. If you do not have hundreds of dollars to spend on a commercial compiler, then you can use one of the free compilers available on the Web.

We will start at the beginning with an extremely simple C program and build up from there. I will assume that you are using the UNIX command line and gcc as your environment for these examples; if you are not, all of the code will still work fine—you will simply need to understand

and use whatever compiler you have available.

2. The Simplest C Program

The best way to get started with C is to write, compile, and execute a simple program. Now, let's start with the simplest possible C program and use it both to understand the basics of C and the C compilation process. Type the following program into a standard text editor. Then save the program to a file named samp.c. If you leave off .c, you will probably get some sort of error when you compile it, so make sure you remember the .c. Also, make sure that your editor does not automatically append some extra characters (such as .txt) to the name of the file. Here's the first program:

```
#include<stdio.h>
int main()
{
  printf("This is output from my first program!\n");
  return 0;
}
```

When executed, this program instructs the computer to print out the line "This is output from my first program!"—then the program quits. You can't get much simpler than that!

Position: When you enter this program, position #include so that the pound sign is in column 1 (the far left side). Otherwise, the spacing and indentation can be any way you like it. The spacing and indentation shown above is a good example to follow.

To compile this code, take the following steps:

① On a UNIX machine, type gcc samp.c -o samp (if gcc does not work, try cc). This line invokes the C compiler called gcc, asks it to compile samp.c and asks it to place the executable file it creates under the name samp. To run the program, type samp.

② On a DOS or Windows machine using DJGPP, at an MS-DOS prompt type gcc samp.c -o samp.exe. This line invokes the C compiler called gcc, asks it to compile samp.c and asks it to place the executable file it creates under the name samp.exe. To run the program, type samp.

③ If you are working with some other compiler or development system, read and follow the directions for the compiler you are using to compile and execute the program.

You should see the output "This is output from my first program!" when you run the program. If you mistyped the program, neither will it compile, nor will it run. You should edit it again and see where you went wrong in your typing. Fix the error and try again.

Let's walk through this program and start to see what the different lines are doing.

- This C program starts with #include<stdio.h>. This line includes the "standard I/O library" into your program. The standard I/O library lets you read input from the keyboard(called "standard in"), write output to the screen (called "standard out"), process text files stored on the disk, and so on. It is an extremely useful library. C has a large number of standard libraries like stdio, including string, time and math libraries. A library is simply a package of code that someone else has written to make your life

easier.

- The line int main() declares the main function. Every C program must have a function named main somewhere in the code. At run time, program execution starts at the first line of the main function.
- In C, the "{" and "}" symbols mark the beginning and end of a block of code. In this case, the block of code making up the main function contains two lines.
- The printf statement in C allows you to send output to standard out (for us, the screen). The portion in quotes is called the format string and describes how the data is to be formatted when printed. The format string can contain string literals, symbols for carriage returns (\n), and operators as placeholders for variables. If you are using UNIX, you can type man 3 printf to get complete documentation for the printf function. If not, see the documentation included with your compiler for details about the printf function.
- The return 0; line causes the function to return an error code of 0 (not error) to the shell that started execution.

3．Variables

As a programmer, you will frequently want your program to "remember" a value. For example, if your program requests a value from the user, or if it calculates a value, you will want to remember it somewhere so you can use it later. The way your program remembers things is by using variables. For example:

```
int b;
```

This line says, "I want to create a space called b that is able to hold one integer value." A variable has a name (in this case, b) and a type (in this case, int, an integer). You can store a value in b by saying something like:

```
b=5;
```

You can use the value in b by saying something like:

```
printf("%d",b);
```

In C, there are several standard types for variables:

- int—integer (whole number) values.
- float—floating point values.
- char—single character values (such as "m" or "Z").

4．Printf

The printf statement allows you to send output to standard out. For us, standard out is generally the screen. Here is another program that will help you learn more about printf:

```
#include<stdio.h>
int main()
{
   int a,b,c;
   a=5;
```

```
    b=7;
    c=a+b;
    printf("%d+%d=%d\n",a,b,c);
    return 0;
}
```

Type this program into a file and save it as add.c. Compile it with the line gcc add.c –o add and then run it by typing add (or ./add). You will see the line "5+7=12" as output.

Here is an explanation of the different lines in this program:

- The line int a,b,c; declares three integer variables named a,b and c.
- The next line initializes the variable named a to the value 5.
- The next line sets b to 7.
- The next line adds a and b and "assigns" the result to c. The computer adds the value in a (5) to the value in b (7) to form the result 12, and then places that new value (12) into the variable c. The variable c is assigned the value 12. For this reason, the = in this line is called "the assignment operator."
- The printf statement then prints the line "5+7=12." The %d placeholders in the printf statement act as placeholders for values. There are three %d placeholders, and at the end of the printf line there are the names for three variables: a,b and c. C matches up the first %d with a and substitutes 5 there. It matches the second %d with b and substitutes 7. It matches the third %d with c and substitutes 12.

Then it prints the completed line to the screen: 5+7=12. The +, the = and the spacing are a part of the format line and get embedded automatically between the %d operators as specified by the programmer.

New Words and Expressions

leverage *n.* 杠杆作用，影响力
binary *adj.* 双重的，双的；二进制的
subscript *adj.* 写在下面的 *n.* 脚注，下标，下角数码
modularization *n.* 模块化
executable *adj.* 可执行的，可实行的，可以做成的
integer *n.* 整数；完整的东西
automatically *adv.* 自动地；机械地

Exercises to the Text

1．Translate the following words and phrases into English.

（1）系统编程语言 （2）中级语言 （3）高级语言 （4）机器语言 （5）汇编语言 （6）系统软件 （7）免费编译器 （8）间距和缩进 （9）标准输入输出库 （10）标准类型

2．Translate the following paragraphs into Chinese.

(1) C has been around for several decades and has won widespread acceptance because it gives programmers maximum control and efficiency. If you are programmer, or if you are

interested in becoming a programmer, there are a couple of benefits you gain from learning C.

(2) There are some composite symbols with special meanings: for example "&&" means "and then" and "||" means "or else". "==" is used for equality to avoid confusion with "=" in assignments, and "!=" is used for inequality.

(3) C is what is called a compiled language. This means that once you write your C program, you must run it through a C compiler to turn your program into an executable that the computer can run (execute). The C program is the human-readable form, while the executable that comes out of the compiler is the machine-readable and executable form. What this means is that to write and run a C program, you must have access to a C compiler.

(4) The standard I/O library lets you read input from the keyboard(called "standard in"), write output to the screen (called "standard out"), process text files stored on the disk, and so on. It is an extremely useful library. C has a large number of standard libraries like stdio, including string, time and math libraries. A library is simply a package of code that someone else has written to make your life easier.

Text 3: C++

C++ is a general-purpose programming language with high-level and low-level capabilities. It is a statically typed, free-form, multi-paradigm, usually compiled language supporting procedural programming, data abstraction, object-oriented programming, and generic programming.

1．Origins of the C++ Language

The C++ programming languages can be thought of as the C Programming language with classes (and other modern features added). The C programming language was developed by Dennis Ritchie of AT&T Bell Laboratories in the 1970s. It was first used for writing and maintaining the UNIX operating system. (Up until that time, UNIX systems programs were written either in assembly language or in language called B, a language developed by Ken Thompson, the originator of UNIX.) C is a general-purpose language that can be used for writing any sort of program, but its success and popularity are closely tied to the UNIX operating system. If you wanted to maintain your UNIX system, you need to use C. C and UNIX fit together so well that soon not just systems programs but almost all commercial programs that ran under UNIX were written in the C language. C became so popular that versions of the language were written for other popular operating systems; its use is thus not limited to computers that use UNIX. However, despite its popularity, C was not without its shortcomings.

The C language is peculiar because it is a high-level language with many of the features of a low-lever language. C is somewhere in between the two extremes of a very high-level language and a low-level language, and therein lies both its strengths and its weakness. Like (low-level) assembly language, C language programs can directly manipulate the computer's memory. On the other hand, C has the features of a high-level language, which makes it easier to read and

write than assembly language. This makes C an excellent choice for writing systems programs, but for other programs (and in some sense even for systems programs) C is not as easy to understand as other languages; also, it does not have as many automatic checks as some other high-level languages.

To overcome these and other shortcomings of C, Bjarne Stroustrup of AT&T Bell Laboratories developed C++ in the early 1980s. Stroustrup designed C++ to be a better C. Most of C is a subset of C++, and so most C programs are also C++ programs. (The reverse is not true; many C++ programs are definitely not C programs.) The basic syntax and semantics of C and C++ are the same. If you are familiar with C, you can program in C++ immediately. C++ has the same types, operators, and other facilities defined in C that usually correspond directly to computer architecture. Unlike C, C++ has facilities for classes and so can be used for object-oriented programming.

2．C++ and Object-Oriented Programming

Object-oriented programming is a programming technique that allows you to view concepts as a variety of objects. By using objects, you can represent the tasks that are to be performed, their interaction, and any given conditions that must be observed. A data structure often forms the basis of an object; thus, in C or C++, the struct type can form an elementary object. Communicating with objects can be done through the use of messages. Using messages is similar to calling a function in a procedure-oriented program. When an object receives a message, methods contained within the object respond. Methods are similar to the functions of procedure-oriented programming. However, methods are part of an object.

The main characteristics of Object-oriented programming (OOP) are encapsulation, inheritance, and polymorphism. Encapsulation is a form of information hiding or abstraction. Inheritance has to do with writing reusable code. Polymorphism refers to a way that a single name can have multiple meanings in the context of inheritance. C++ accommodates OOP by providing classes, a kind of data type combining both data and algorithms. C++ is not what some authorities would call a “pure OOP language.” C++ tempers its OOP features with concerns for efficiency and what some might call “practicality.” This combination has mad C++ currently the most widely used OOP language, although not all of its usage strictly follows the OOP philosophy.

3．The Character of C++

C++ allows the programmer to create classes, which are somewhat similar to C structures. In C++, it can be assigned methods, functions associated to it, of various prototypes, which can access and operate within the class, somewhat like C functions often operate on a supplied handler pointer. The C++ class is an extension of the C language structure. Because the only difference between a structure and a class is that structure members have public access by default and a class member has private access by default, you can use the keywords class or struct to define equivalent classes.

The C++ class is an extension of the C and C++ struct type and forms the required abstract

data type for object-oriented programming. The class can contain closely related items that share attributes. Stated more formally, an object is simply an instance of a class.

Ultimately, there should emerge class libraries containing many object types. You could use instances of those object types to piece together program code.

Typically, an object's description is part of a C++ class and includes a description of the object's internal structure, how the object relates with other objects, and some form of protection that isolates the functional details of the object from outside the class. The C++ class structure does all of this.

In a C++ class, you control functional details of the object by using private, public, or protected descriptors. In object-oriented programming, the public section is typically used for the interface information (methods) that makes the class reusable across applications. If data or methods are contained in the public section, they are available outside the class. The private section of a class limits the availability of data or methods to the class itself. A protected section containing data or methods is limited to the class and any derived subclasses.

C++'s connection to the C language gives it a more traditional look than newer object-oriented languages, yet it has more powerful abstraction mechanisms than many other currently popular languages. C++ has a template facility that allows for full and direct implementation of algorithm abstraction. C++ templates allow you to code using parameters for types. The newest C++ standard, and most C++ compilers, allow multiple namespaces to accommodate more reuse of class and function names. The exception handling facilities in C++ are similar to what you would find in other programming languages. Memory management in C++ is similar to that in C. The programmer must allocate his or her own memory and handle his or her own garbage collection. Most compilers will allow you to do C-style memory management in C++, since C is essentially a subset of C++. However, C++ also has its own syntax for a C++ style of memory management, and you are advised to use the C++ style of memory management when coding in C++.

Inheritance in object-oriented programming allows a class to inherit properties from a class of objects. The parent class serves as a pattern for the derived class and can be altered in several ways. If an object inherits its attributes from multiple parents, it is called multiple inheritance. Inheritance is an important concept since it allows reuse of a class definition without requiring major code changes. Inheritance encourages the reuse of code since child classes are extensions of parent classes.

Another important object-oriented concept that relates to the class hierarchy is that common messages can be sent to the parent class objects and all derived subclass objects. In formal terms, this is called polymorphism.

Polymorphism allows each subclass object to respond to the message format in a manner appropriate to its definition. Imagine a class hierarchy for gathering data. The parent class might be responsible for gathering the name, social security number, occupation, and number of years of employment for an individual. You could then use child classes to decide what additional

information would be added based on occupation. In one case a supervisory position might include yearly salary, while in another case a sales position might include an hourly rate and commission information. Thus, the parent class gathers general information common to all child classes while the child classes gather additional information relating to specific job descriptions. Polymorphism allows a common data-gathering message to be sent to each class. Both the parent and child classes respond in an appropriate manner to the message.

Polymorphism gives objects the ability to responds to messages from routines when the object's exact type is not known. In C++ this ability is a result of late binding. With late binding, the addresses are determined dynamically at run time, rather than statically at compile time, as in traditional compiled languages. This static method is often called early binding. Function names are replaced with memory addresses. You accomplish late binding by using virtual functions. Virtual functions are defined in the parent class when subsequent derived classes will overload the function by redefining the function's implementation.

Virtual functions utilize a table for address information. The table is initialized at run time by using a constructor. A constructor is invokes whenever an object of its class is created. The job of the constructor here is to link the virtual function with the table of address information. During the compile operation, the address of the virtual function is not known; rather, it is given the position in the table of addresses that will contain the address for the function.

4. The C++ Program

All procedure-like entities are called functions in C++. Things that are called procedures, methods, functions, or subprograms in other languages are all called functions in C++. A C++ program is basically just a function called main; when you run a program, the run-time system automatically invokes the function named main. Other C++ terminology is pretty much the same as most other programming languages. Here's the C++ program.

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5    int numberOfLanguage;
6    cout<<"Hello reader.\n"
7        <<"Welcome to C++.\n";
8    cout<<"How many programming languages have you used?";
9    cin>>numberOfLanguages;
10    if(numberOfLanguages<1)
11      cout<<"Read the preface. You may prefer\n"
12         <<"a more elementary book by the same author.\n";
13    else
14      cout<<"Enjoy the book.\n";
15  return 0;
16  }
```

A C++ program is really a function definition for a function named main. When the program is run, the function named main is invoked. The body of the function main is enclosed in braces, {}. When the program is run, the statements in the braces are executed. Here are two possible screen displays that might be generated when a user runs the program.

DIALOGUE 1

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 0← User types in 0 on the keyboard.
Read the preface. You may prefer a more elementary book by the same author.
```

DIALOGUE 2

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 1← User types in 1 on the keyboard.
Enjoy the book
```

Variable declarations in C++ are similar to what they are in other programming languages. The fifth line declares the variable numberOfLanguages. The type int is one of the C++ types for whole numbers (integers).

If you have not programmed in C++ before, then the use of cin and cout for console I/O is likely to be new to you. But the general idea can be observed in this sample program. For example, consider the eighth line and the ninth lines. The eighth line outputs the text within the quotation marks to the screen. The ninth line reads in a number that the user enters at the keyboard and sets the value of the variable numberOfLanguages to this number.

The eleventh line and the twelfth output two strings instead of just one string. The symbolism \n is the new line character, which instructs the computer to start a new line of output.

New Words and Expressions

architecture *n.* 建筑学
capability *n.* 能力
enhance *v.* 改善，提高，增进
semantics *n.* 语义学，语义论
concept *n.* 概念，观念，思想
interaction *n.* 交互作用，相互作用
extension *n.* 伸展，扩大，延长，延期，电话分机
equivalent *adj.* 等价的，相等的，同意义的 *n.* 等价物，相等物
description *n.* 描述，描写，说明书，类型
subclass *n.* 子类，亚类
hierarchy *n.* 层级，等级制度
inherit *vt.* 继承，遗传而得 *vi.* 成为继承人
inheritance *n.* 遗产，继承，遗传
hourly *adv.* 频繁地，随时，每小时地 *adj.* 每小时的，以钟点计算的，频繁的

Exercises to the Text

1．Translate the following words and phrases into English.

（1）编程语言 （2）面向对象程序设计 （3）抽象机制 （4）代码重用 （5）接口信息 （6）地址信息表 （7）虚函数 （8）附加信息 （9）函数定义 （10）变量声明

2．Translate the following paragraphs into Chinese.

(1) C++ is a general-purpose programming language with high-level and low-level capabilities. It is a statically typed, free-form, multi-paradigm, usually compiled language supporting procedural programming, data abstraction, object-oriented programming, and generic programming.

(2) C is a general-purpose language that can be used for writing any sort of program, but its success and popularity are closely tied to the UNIX operating system. If you wanted to maintain your UNIX system, you need to use C. C and UNIX fit together so well that soon not just systems programs but almost all commercial programs that ran under UNIX were written in the C language.

(3) Object-oriented programming is a programming technique that allows you to view concepts as a variety of objects. By using objects, you can represent the tasks that are to be performed, their interaction, and any given conditions that must be observed.

(4) Polymorphism allows each subclass object to respond to the message format in a manner appropriate to its definition. Imagine a class hierarchy for gathering data. The parent class might be responsible for gathering the name, social security number, occupation, and number of years of employment for an individual. You could then use child classes to decide what additional information would be added based on occupation.

(5) Virtual functions utilize a table for address information. The table is initialized at run time by using a constructor. A constructor is invokes whenever an object of its class is created. The job of the constructor here is to link the virtual function with the table of address information.

Text 4: Java

Java-has become enormously popular. Java's rapid rise and wide acceptance can be traced to its design and programming feature, particularly its promise that you can write a program once and run anywhere. As stated in the Java-language white paper by Sun, Java simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic.

What Is Java?

Java is a major departure from the HTML coding that makes up most Web pages. Sitting atop markup languages such as HTML and XML, Java is an object-oriented, network-friendly high-level programming language that allows programmers to build applications that can run on

almost any operating system. With Java, big applications programs can be broken into mini-applications, or “applets”, that can be downloaded off the Internet and run on any computer. Moreover, Java enables a Web page to deliver, along with visual content, applets that when downloaded can make Web pages interactive.

Some microcomputers include special Java microprocessors designed to run Java software directly. However, Java is not compatible with many existing microprocessors, such as those from Intel and Motorola. For this reason, these users need to use a small “interpreter” program, called a Java Virtual Machine that translates a Java program into a language that any computer or operating system can understand. They also need a Java-capable browser in order to view Java special effects on the Web.

Java development programs are available for programmers. In addition, Java software packages—such as ActionLine, Activator Pro, AppletAce, and Mojo-give nonprogrammers the ability to add multimedia effects to their Web pages, by producing applets that any Java-equipped browser can view. Such packages can be used by anyone who understands multimedia file formats and is willing to experiment with menu options.

Java is the latest, flashiest object-oriented language. One especially powerful feature of OOP (object-oriented programming) languages is a property which is known as inheritance. Inheritance allows an object to take on the characteristics and functions of other objects to which it is functionally connected. Programmers connect objects by grouping them together in different classes and by grouping the classes into hierarchies. These classes and hierarchies allow programmers to define the characteristics and functions of objects without needing to repeat source code. Thus, using OOP languages can greatly reduce the time it takes for a programmer to write an application, and also reduce the size of the program. OOP languages are flexible and adaptable, so programs or parts of programs can be used for more than one task. Programs written with OOP languages are generally shorter in length and contain fewer bugs, or mistakes, than those written with non-OOP languages.

Java has taken the software world by storm due to its close ties with the Internet and Web browsers. It is designed as a portable language that can run on any Web-enabled computer via that computer’s Web browser. As such, it offers great promise as the standard Internet and Intranet programming language.

It has the syntax of C++, making it easy (or difficult) to learn, depending on your experience. But it has improved on C++ in some important areas. For one thing, it has no pointers, which are low-level programming constructs that make for error-prone programs. It has garbage collection, a feature that frees the programmer from explicitly allocating and de-allocating memory. It also runs on a virtual machine, which is software built into a Web browser that executes the same standard compiled Java bytecodes no matter the type of computer.

Java development tools are being rapidly deployed, and are available from such major software companies as IBM, Microsoft, and Symantec.

The Characteristics of Java

1．Java Is Simple

No language is simple, but Java is a bit easier than the popular object-oriented programming language C++, which was the dominant software-development language before Java. Java is partially modeled on C++, but greatly simplified and improved. For instance, pointers and multiple inheritances often make programming complicated. Java replaces the multiple inheritances in C++ with a simple language construct called an interface, and eliminates pointers.

Java uses automatic memory allocation and garbage collection, whereas C++ requires the programmer to allocate memory and collect garbage. Also, the number of language constructs is small for such a powerful language. The clean syntax makes Java programs easy to write and read. Some people refer to Java as "C++−−" because it is like C++ but with more functionality and fewer negative aspects.

2．Java Is Object-Oriented

Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

Software systems developed using procedural programming languages are based on the paradigm of procedures. Object-oriented programming models the real world in terms of objects. Everything in the world can be modeled as an object. A circle is an object, a person is an object, and a Windows icon is an object. Even a loan can be perceived as an object. A Java program is object-oriented because programming in Java is centered on creating objects, manipulating objects, and making objects work together.

One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism. For years, object-oriented technology was perceived as elitist, requiring a substantial investment in training and infrastructure. Java has helped object-oriented technology enter the mainstream of computing. Its simple, clean syntax makes programs easy to write and read. Java programs are quite expressive in terms of designing and developing applications.

3．Java Is Distributed

Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network program is like sending and receiving data to and from a file.

4．Java Is Interpreted

You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any

machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).

Most compilers, including C++ compilers, translate programs in a high-level language to machine code. The code can only run on the native machine. If you run the program on other machines, it has to be recompiled on the native machine. For instance, if you compile a C++ program in Windows, the executable code generated by the compiler can only run on the Windows platform. With Java, you compile the source code once, and the bytecode generated by a Java compiler can run on any platform with a Java interpreter. The Java interpreter translates the bytecode into the machine language of the target machine.

5．Java Is Robust

Robust means reliable. No programming language can ensure complete reliability. Java puts a lot of emphasis on early checking for possible errors, because Java compilers can detect many problems that would first show up at execution time in other languages. Java has eliminated certain types of error-prone programming constructs found in other languages. It does not support pointers, for example, thereby eliminating the possibility of overwriting memory and corrupting data.

Java has a runtime exception-handling feature to provide programming support for robustness. Java forces the programmer to write the code to deal with exceptions. Java can catch and respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs.

6．Java Is Secure

As an Internet programming language, Java is used in a networked and distributed environment. If you download a Java applet (a special kind of program) and run it on your computer, it will not damage your system because Java implements several security mechanisms to protect your system against harm caused by stray programs. The security is based on the premise that nothing should be trusted.

7．Java Is Architecture-Neutral

Java is interpreted. This feature enables Java to be architecture-neutral, or to use an alternative term, platform-independent. With a Java Virtual Machine (JVM), you can write one program that will run on any platform.

Java’s initial success stemmed from its Web-programming capability. You can run Java applets from a Web browser, but Java is for more than just writing Web applets. You can also run standalone Java applications directly from operating systems, using a Java interpreter. Today, software vendors usually develop multiple versions of the same product to run on different platforms. Using Java, developers need to write only one version that can run on every platform.

8．Java Is Portable

Because Java is architecture-neutral, Java programs are portable. They can be run on any platform, without being recompiled. Moreover, there are no platform-specific features in the Java language. In some languages, such as Ada, the largest integer varies on different platforms. But

in Java, the range of the integer is the same on every platform, as is the behavior of arithmetic. The fixed range of the numbers makes the program portable.

The Java environment is portable to new hardware and operating systems. In fact, the Java compiler itself is written in Java.

9．Java's Performance

Java's performance is sometimes criticized. The execution of the bytecode is never as fast as it would be with a compiled language, such as C++. Because Java is interpreted, the bytecode is not directly executed by the system, but is run through the interpreter. However, its speed is more than adequate for most interactive applications, where the CPU is often idle, waiting for input or for data from other sources.

CPU speed has increased dramatically in the past few years, and this trend will continue. There are many ways to improve performance. Users of the earlier Sun Java Virtual Machine certainly noticed that Java was slow. However, the new JVM is significantly faster. The new JVM uses the technology known as just-in-time compilation. It compiles bytecode into native machine code, stores the native code, and reinvokes the native code when its bytecode is executed. Sun recently developed the Java HotSpot Performance Engine, which includes a compiler for optimizing the frequently used code. The HotSpot Performance Engine can be plugged into a JVM to dramatically boost its performance.

10．Java Is Multithreaded

Multithreading is a program's capability to perform several tasks simultaneously. Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading. Multithreading is particularly useful in graphical user interface (GUI) and network programming. In GUI programming, there are many things going on at the same time. A user can listen to an audio recording while surfing a Web page. In network programming, a server can serve multiple clients at the same time. Multithreading is a necessity in multimedia and network programming.

11．Java Is Dynamic

Java was designed to adapt to an evolving environment. New class can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

What Can Java Technology Do?

The most common types of programs written the Java programming are applets and application. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled. At the beginning of this trail is an applet that displays an animation of the Java technology mascot, Duke, waving at you.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are web servers, proxy servers, mail servers, and print servers. Another specialized program is a Servlet. A Servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, Servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- The essentials—Objects, strings, threads, numbers, input and output data structures, system properties, date and time, and so on.
- Applets—The set of conventions used by applets.
- Networking—URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.
- Internationalization—Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- Security—Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- Software components—Known as JavaBeans, can plug; into existing component architectures.
- Object serialization—Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- Java Database Connectivity (JDBC)—Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more.

Note: The Java 2 SDK, Standard Edition v. 1.3. The Java 2 Runtime Environment (JRE) consists of the virtual machine, the Java platform core classes, and supporting files. The Java 2 SDX includes the JRE and development tools such as compilers and debuggers.

New Words and Expressions

incorporated　*adj*. 组成公司的，合成一体的
enormously　*adv.* 巨大地，庞大地，非常地，在极大程度上
architecture　*n*. 建筑学 建筑式样，建筑风格
multithreaded　*adj*. 多线程的，多重线串的
interactive　*adj.* 相互作用的，交互式的

microcomputer *n.* 微型计算机，微电脑
inheritance *n.* 继承（性）
memory allocation 存储器分配
garbage *n.* 垃圾，无用信息（或单元、数据）
garbage collection 垃圾收集，无用（存储）单元收集
functionality *n.* 功能性
bytecode *n.* 字节码，字节代码，位元码
inherent *adj.* 内在的；固有的
allocation *n.* 分配，配置，安置
paradigm *n.* 范例，样式；范式
modularity *n.* 模块性，模块化
clarity *n.* 明晰，明确（性）
reusability *n.* 可重用性，可重复使用性
encapsulation *n.* 封装，密封；压缩
polymorphism *n.* 多态性，多形性
elitist *adj.* 杰出人物的；上等的，高级的
infrastructure *n.* 基础结构，基础设施
mainstream *n.* 主流
expressive *adj.* 表现的；富于表现力的
integrate *v.* （使）成为一体（或结合、合并）
compiler *n.* 编辑者，汇编者，编译器，编译程序
platform *n.* 月台，站台，讲台，平台
arithmetic *n.* 算术，算法
bytecode *n.* 字节码，字节代码
prone *adj.* 有……倾向的，易于……的
overwrite *v.* 改写，重写
corrupt *v.* 毁坏，损坏，破坏
runtime *n.* 运行时刻，运行（时）期
exception *n.* 异常（事件），例外
terminate *v.* （使）终止
stray *n.* 迷路的；杂散的；寄生的
premise *n.* 假设；前提
Web browser 万维网浏览器
standalone *adj.* （相对于操作系统而言）独立的
integer *n.* 整数
optimizing *n.* 优化，最佳化 *adj.* 最佳的
multithreading *n.* 多线程，多线索
extension *n.* 扩展，扩张，延长
internationalization *n.* 国际化
animation *n.* 活泼，生气，卡通片绘制

Exercises to the Text

1．Translate the following words and phrases into English.

（1）存储器分配 （2）面向对象程序设计 （3）机器码 （4）万维网浏览器 （5）可执行代码 （6）运行期异常处理 （7）Java 虚拟机 （8）即时编译 （9）可移植应用软件 （10）本机代码

2．Translate the following paragraphs into Chinese.

(1) Java is a major departure from the HTML coding that makes up most Web pages. Sitting atop markup languages such as HTML and XML, Java is an object-oriented, network-friendly high-level programming language that allows programmers to build applications that can run on almost any operating system.

(2) Java has taken the software world by storm due to its close ties with the Internet and Web browsers. It is designed as a portable language that can run on any Web-enabled computer via that computer's Web browser. As such, it offers great promise as the standard Internet and Intranet programming language.

(3) No language is simple, but Java is a bit easier than the popular object-oriented programming language C++, which was the dominant software-development language before Java. Java is partially modeled on C++, but greatly simplified and improved. For instance, pointers and multiple inheritances often make programming complicated. Java replaces the multiple inheritances in C++ with a simple language construct called an interface, and eliminates pointers.

(4) Multithreading is particularly useful in graphical user interface (GUI) and network programming. In GUI programming, there are many things going on at the same time. A user can listen to an audio recording while surfing a Web page. In network programming, a server can serve multiple clients at the same time. Multithreading is a necessity in multimedia and network programming.

(5) Java was designed to adapt to an evolving environment. New class can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

Text 5: ActionScript Basics

Where to Place ActionScript Code?

If you have a new ActionScript project, do you know where to put the code for it to execute properly? The answer is, place ActionScript code in the constructor and additional methods of the class.

In ActionScript 1.0 and 2.0, you had many choices as to where to place your code: on the timeline, on buttons and movie clips, on the timeline of movie clips, in external .as files referenced with #include, or as external class files. ActionScript 3.0 is completely class-based, so

all code must be placed in methods of your project's classes.

When you create a new ActionScript project, the main class is automatically created, and opened in the Code view. It should look something like this:

```
package {
    import flash.display.Sprite;
    public class ExampleApplication extends Sprite
    {
        public function ExampleApplication()
        {
        }
    }
}
```

Even if you are familiar with classes in ActionScript 2.0, there are some new things here.

The first thing you'll notice is the word package at the top of the code listing. Packages are used to group classes of associated functionality together. In ActionScript 2.0, packages were inferred through the directory structure used to hold the class files. In ActionScript 3.0, however, you must explicitly specify packages. For example, you could have a package of utility classes. This would be declared like so:

```
package com.as3cb.utils {
}
```

If you don't specify a package name, your class is created in the default, top-level package. You should still include the package keyword and braces.

Next, place any import statements. Importing a class makes that class available to the code in the file and sets up a shortcut so you don't have to type the full package name every time you want to refer to that class. For example, you can use the following import statement:

```
import com.as3cb.utils.StringUtils;
```

Thereafter you can refer to the StringUtils class directly without typing the rest of the path. As shown in the earlier example, you will need to import the Sprite class from the flash.display package, as the default class extends the Sprite class.

Next up is the main class, ExampleApplication. You might notice the keyword public in front of the class definition. Although you can't have private classes within a package, you should label the class public. Note that the main class extends Sprite. Also, a .swf itself is a type of sprite or movie clip, which is why you can load a .swf into another .swf and largely treat it as if it were just another nested sprite or movie clip. This main class represents the .swf as a whole, so it should extend the Sprite class or any class that extends the Sprite class (such as MovieClip).

Finally, there is a public function (or method, in class terminology) with the same name as the class itself. This makes it a constructor. A class's constructor is automatically run as soon as an instance of the class is created. In this case, it is executed as soon as the .swf is loaded into the

Flash player. So where do you put your code to get it to execute? Generally, you start out by putting some code in the constructor method. Here's a very simple example that just draws a bunch of random lines to the screen:

```
package {
    import flash.display.Sprite;
    public class ExampleApplication extends Sprite {
        public function ExampleApplication() {
            graphics.lineStyle(1, 0, 1);
            for(var i:int=0;i<100;i++) {
                graphics.lineTo(Math.random() * 400, Math.random() * 400);
            }
        }
    }
}
```

Save and run the application. Your browser should open the resulting HTML file and display the .swf with 100 random lines in it. As you can see, the constructor was executed as soon as the file was loaded into the player.

In practice, you usually want to keep code in the constructor to a bare minimum. Ideally the constructor would just contain a call to another method that initializes the application.

For beginners, now that you know where to enter code, here is quick primer on terminology. These definitions are briefly stated and intended to orient people who have never programmed before.

- Variables

Variables are convenient placeholders for data in your code, and you can name them anything you'd like, provided the name isn't already reserved by ActionScript and the name starts with a letter, underscore, or dollar sign (but not a number). The help files installed with Flex Builder 2 contain a list of reserved words. Variables are convenient for holding interim information, such as a sum of numbers, or to refer to something, such as a text field or sprite. Variables are declared with the var keyword the first time they are used in a script. You can assign a value to a variable using an equal sign (=), which is also known as the assignment operator. If a variable is declared outside a class method, it is a class variable. Class variables, or properties, can have access modifiers, public, private, protected, or internal. A private variable can only be accessed from within the class itself, whereas public variables can be accessed by objects of another class. Protected variables can be accessed from an instance of the class or an instance of any subclass, and internal variables can be accessed by any class within the same package. If no access modifier is specified, it defaults to internal.

- Functions

Functions are blocks of code that do something. You can call or invoke a function (that is, execute it) by using its name. When a function is part of a class, it is referred to as a method of

the class. Methods can use all the same modifiers as properties.

- Scope

A variable's scope describes when and where the variable can be manipulated by the code in a movie. Scope defines a variable's life span and its accessibility to other blocks of code in a script. Scope determines how long a variable exists and from where in the code you can set or retrieve the variable's value. A function's scope determines where and when the function is accessible to other blocks of code.

- Event handler

A handler is a function or method that is executed in response to some event such as a mouseclick, a keystroke, or the movement of the playhead in the timeline.

- Objects and classes

An object is something you can manipulate programmatically in ActionScript, such as a sprite. There are other types of objects, such as those used to manipulate colors, dates, and text fields. Objects are instances of classes, which means that a class is a template for creating objects and an object is a particular instance of that class. If you get confused, think of it in biological terms: you can consider yourself an object (instance) that belongs to the general class known as humans.

- Methods

A method is a function associated with an object that operates on the object. For example, a text field object's replaceSelectedText( ) method can be used to replace the selected text in the field.

- Properties

A property is an attribute of an object, which can be read and/or set. For example, a sprite's horizontal location is specified by its x property, which can be both tested and set. On the other hand, a text field's length property, which indicates the number of characters in the field, can be tested but cannot be set directly (it can be affected indirectly, however, by adding or removing text from the field).

- Statements

ActionScript commands are entered as a series of one or more statements. A statement might tell the playhead to jump to a particular frame, or it might change the size of a sprite. Most ActionScript statements are terminated with a semicolon (;).

- Comments

Comments are notes within code that are intended for other humans and ignored by Flash. In ActionScript, single-line comments begin with // and terminate automatically at the end of the current line. Multiline comments begin with /* and are terminated with */.

- Interpreter

The ActionScript interpreter is that portion of the Flash Player that examines your code and attempts to understand and execute it. Following ActionScript's strict rules of grammar ensures that the interpreter can easily understand your code. If the interpreter encounters an error, it often

fails silently, simply refusing to execute the code rather than generating a specific error message.

- Handling Events

If you want to have some code repeatedly execute, add a listener to the enterFrame event and assign a method as a handler.

In ActionScript 2.0 handling the enterFrame event was quite simple. You just had to create a timeline function called onEnterFrame and it was automatically called each time a new frame began. In ActionScript 3.0, you have much more control over the various events in a .swf, but a little more work is required to access them.

If you are familiar with the EventDispatcher class from ActionScript 2.0, you should be right at home with ActionScript 3.0's method of handling events. In fact, EventDispatcher has graduated from being an externally defined class to being the base class for all interactive objects, such as sprites.

To respond to the enterFrame event, you have to tell your application to listen for that event and specify which method you want to be called when the event occurs. This is done with the addEventListener method, which is defined as follows:

```
addEventListener(type:String, listener:Function)
```

The type parameter is the type of event you want to listen to. In this case, it would be the string, "enterFrame". However, using string literals like that opens your code to errors that the compiler cannot catch. If you accidentally typed "enterFrame", for example, your application would simply listen for an "enterFrame" event. To guard against this, it is recommended that you use the static properties of the Event class. You should already have the Event class imported, so you can call the addEventListener method as follows:

```
addEventListener(Event.ENTER_FRAME, onEnterFrame);
```

Now if you accidentally typed Event.ENTER_FRAME, the compiler would complain that such a property did not exist.

The second parameter, onEnterFrame, refers to another method in the class. Note, that in ActionScript 3.0, there is no requirement that this method be named onEnterFrame. However, naming event handling methods on plus the event name is a common convention. This method gets passed an instance of the Event class when it is called. Therefore, you'll need to import that class and define the method so it accepts an event object:

```
import flash.events.Event;
private function onEnterFrame(event:Event) {
}
```

The event object contains information regarding the event that may be useful in handling it. Even if you don't use it, you should still set your handler up to accept it. If you are familiar with the ActionScript 2.0 version of EventDispatcher, you'll see a difference in implementation here. In the earlier version, there was an issue with the scope of the function used to handle the event, which often required the use of the Delegate class to correct. In ActionScript 3.0, the scope of

the handling method remains the class of which it is a method, so there is no necessity to use Delegate to correct scope issues.

Here is a simple application that draws successive random lines, using all the concepts discussed in this recipe:

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
        public class ExampleApplication extends Sprite {
            public function ExampleApplication() {
            graphics.lineStyle(1, 0, 1);
            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void {
            graphics.lineTo(Math.random() * 400, Math.random() * 400);
        }
    }
}
```

New Words and Expressions

terminology *n.* 专门用语，术语
interim *adj.* 暂时的，临时的，间歇的 *n.* 过渡时期
invoke *vt.* 援引，援用；行使（权利等）
modifier *n.* [语言学]修饰语
retrieve *vt.* 取回，挽回，弥补，恢复，补偿，回忆，检索
manipulate *vt.* 熟练控制（操作）
semicolon *n.* 分号
successive *adj.* 连续的，相继的
regarding *prep.*（表示论及）关于；至于；就……而论
version *n.* 版本，形式
terminate *vt.&vi.* 结束；使终结
execute *vt.* 实行，实施，执行；完成，实现，履行
underscore *vt.* 在……下画线；强调
nested *adj.* 嵌套的
explicitly *adv.* 明白地，明确地

Exercises to the Text

1．Translate the following words and phrases into English.

（1）变量 （2）函数 （3）作用域 （4）事件处理器 （5）对象和类 （6）方法 （7）属性 （8）语句 （9）注释 （10）解释器

2．Translate the following paragraphs into Chinese.

(1) Variables are convenient placeholders for data in your code, and you can name them anything you'd like, provided the name isn't already reserved by ActionScript and the name starts with a letter, underscore, or dollar sign (but not a number).

(2) A property is an attribute of an object, which can be read and/or set. For example, a sprite's horizontal location is specified by its x property, which can be both tested and set. On the other hand, a text field's length property, which indicates the number of characters in the field, can be tested but cannot be set directly.

(3) In the earlier version, there was an issue with the scope of the function used to handle the event, which often required the use of the Delegate class to correct. In ActionScript 3.0, the scope of the handling method remains the class of which it is a method, so there is no necessity to use Delegate.

(4) If you are familiar with the EventDispatcher class from ActionScript 2.0, you should be right at home with ActionScript 3.0's method of handling events. In fact, EventDispatcher has graduated from being an externally defined class to being the base class for all interactive objects, such as sprites.