Unit 5

Text A

Computer Programming

Computer programming (often shortened to programming or coding) is the process of writing, testing, and maintaining the source code of computer programs. The source code is written in a programming language. This code may be a modification of existing source or something completely new, and the purpose is to create a program that exhibits the desired behavior. The process of writing source code requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.

Within software engineering, programming (the implementation) is regarded as one phase in a software development process.

In some specialist applications or extreme situations, a program may be written or modified (known as patching) by directly storing the numeric values of the machine code instructions to be executed into memory.

There is an ongoing debate on the extent to which the writing of programs is an art, a craft or an engineering discipline. Good programming is generally considered to be the measured application of all three: expert knowledge informing an elegant, efficient, and maintainable software solution (the criteria for "efficient" and "maintainable" vary considerably). The discipline differs from many other technical professions in that programmers generally do not need to be licensed or pass any standardized (or governmentally regulated) certification tests in order to call themselves "programmers" or even "software engineers".

Another ongoing debate is the extent to which the programming language used in writing programs affects the form that the final program takes. This debate is analogous to that surrounding the Sapir-Whorf hypothesis in linguistics.

1. Programmers

Computer programmers are those who write computer software. Their job usually involves: requirements analysis, specification, software architecture, coding, compilation, software testing,

documentation, integration, maintenance.

2. Programming Languages

Different programming languages support different styles of programming (called programming paradigms). The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Ideally the programming language best suited for the task at hand will be selected. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.

3. Modern Programming

3.1 Algorithmic Complexity

The academic field and engineering practice of computer programming are largely concerned with discovering and implementing the most efficient algorithms for a given class of problem. For this purpose, algorithms are classified into *orders* using so-called Big O notation, O(n), which expresses execution time, memory consumption, or another parameter in terms of the size of an input. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities, and use this knowledge to consider design trade-offs between, for example, memory consumption and performance.

Research in computer programming includes investigation into the unsolved proposition that *P*, the class of algorithms which can be deterministically solved in polynomial time with respect to an input, is not equal to *NP*, the class of algorithms for which no polynomial-time solutions are known. Work has shown that many NP algorithms can be transformed, in polynomial time, into others, such as the Travelling Salesman Problem (TSP), thus establishing a large class of “hard” problems which are for the purposes of analysis.

3.2 Methodologies

The first step in every software development project should be requirements analysis, followed by modeling, implementation, and failure elimination (debugging).

There are a lot of differing approaches for each of those tasks. One approach popular for requirements analysis is Use Case analysis.

Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). The Unified Modeling Language (UML) is a notation used for both OOAD and MDA.

A similar technique used for database design is Entity-Relationship Modeling (ER Modeling).

Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.

Debugging is most often done with IDEs like Visual Studio, and Eclipse. Separate debuggers like gdb are also used.

3.3 Measuring Language Usage

It is very difficult to determine what the most popular of modern programming languages are. Some languages are very popular for particular kinds of applications (e.g. Cobol is still strong in the corporate data center, often on large mainframes, Fortran in engineering applications, and C in embedded applications), while some languages are regularly used to write many different kinds of applications.

Methods of measuring language popularity include: counting the number of job advertisements that mention the language, the number of books teaching the language that are sold (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as Cobol).

3.4 Debugging

Debugging is a very important task for every programmer because an erroneous program is often useless. Languages like C++ and Assembler are very challenging even to expert programmers because of failure modes like Buffer overruns, bad pointers or uninitialized memory. A buffer overrun can damage adjacent memory regions and cause a failure in a totally different program line. Because of those memory issues tools like Valgrind, Purify or Boundschecker are virtually a necessity for modern software development in the C++ language. Languages such as Java, PHP and Python protect the programmer from most of these runtime failure modes, but this may come at the price of a dramatically lower execution speed of the resulting program. This is acceptable for applications where execution speed is determined by other considerations such as database access or file I/O. The exact cost will depend upon specific implementation details. Modern Java virtual machines, for example, use a variety of sophisticated optimizations, including runtime conversion of interpreted instructions to native machine code.

New Words

exhibit	[ig'zibit]	*v.* 展示
expertise	[ˌekspəː'tiːz]	*n.* 专门技术
application	[ˌæpli'keiʃən]	*n.* 应用
phase	[feiz]	*n.* 阶段，时期
patching	['pætʃiŋ]	*n.* 修补，打补丁
ongoing	['ɔngəuiŋ]	*adj.* 不间断的，正在进行的；前进的
debate	[di'beit]	*v. & n.* 争论，辩论
extent	[iks'tent]	*n.* 程度
craft	[krɑːft]	*n.* 工艺，手艺

elegant	[ˈeligənt]	*adj.* 简炼的，简洁的，优雅的，漂亮的
maintainable	[menˈteinəbl]	*adj.* 可维护的
considerably	[kənˈsidərəbəli]	*adv.* 相当地
regulate	[ˈregjuleit]	*vt.* 管制，控制，调节，校准
license	[ˈlaisəns]	*vt.* 发许可证给，批准，许可
licensed	[ˈlaisənst]	*adj.* 得到许可的
linguistics	[liŋˈgwistiks]	*n.* 语言学
compilation	[ˌkɔmpiˈleiʃən]	*n.* 汇编；编辑
documentation	[ˌdɔkjumenˈteiʃən]	*n.* 文件
maintenance	[ˈmeintinəns]	*n.* 维护，保持
suitability	[ˌsjuːtəˈbiləti]	*n.* 合适，适当，相配
availability	[əˌveiləˈbiliti]	*n.* 可用性，有效性，实用性
preference	[ˈprefərəns]	*n.* 喜好，偏爱
ideally	[aiˈdiəli]	*adv.* 理想地，在观念上地
academic	[ˌækəˈdemik]	*adj.* 学院的，理论的
consumption	[kənˈsʌmpʃən]	*n.* 消耗，消费
respective	[risˈpektiv]	*adj.* 分别的，各自的
complexity	[kəmˈpleksiti]	*n.* 复杂性，复杂的事物
investigation	[inˌvestiˈgeiʃən]	*n.* 调查，研究
unsolved	[ˈʌnˈsɔlvd]	*adj.* 未解决的
proposition	[ˌprɔpəˈziʃən]	*n.* 陈述，命题；主张，建议
deterministic	[diˌtəːmiˈnistik]	*adj.* 确定性的，确定的
deterministically	[diˌtəːmiˈnistikəli]	*adv.* 确切地，确定地
polynomial	[ˌpɔliˈnəumiəl]	*adj.* 多项式的
		n. 多项式
methodology	[meθəˈdɔlədʒi]	*n.* 方法
debugging	[diːˈbʌgiŋ]	*n.* 调试
notation	[nəuˈteiʃən]	*n.* 符号
overestimate	[ˈəuvəˈestimeit]	*vt.* 评价过高
		n. 估计得过高，评价得过高
underestimate	[ˈʌndərˈestimeit]	*vt. & n.* 低估
erroneous	[iˈrəuniəs]	*adj.* 错误的，不正确的
useless	[ˈjuːslis]	*adj.* 无用的，无效的，无益的，无价值的
challenging	[ˈtʃælindʒiŋ]	*adj.* 挑战性的
overrun	[ˌəuvəˈrʌn]	*n.* 泛滥，超出限度
initialize	[iˈniʃəlaiz]	*vt.* 初始化

uninitialized	[ʌni'niʃəlaizd]	*adj.* 已初始化的
adjacent	[ə'dʒeisənt]	*adj.* 邻近的，接近的
necessity	[ni'sesiti]	*n.* 必要性，需要，必需品
protect	[prə'tekt]	*vt.* 保护
acceptable	[ək'septəbl]	*adj.* 可接受的，合意的
consideration	[kənsidə'reiʃən]	*n.* 体谅，考虑

Phrases

programming language	程序设计语言
differ from	与……不同，不同于……
formal logic	形式逻辑
regard as …	把……认作
software engineer	软件工程师
be analogous to	类似于，类同于
requirements analysis	需求分析
software architecture	软件构架
programming paradigm	编程范式
be subject to	服从，以……为条件，受……的支配
in terms of	根据，就……而言
be familiar with	熟悉
with respect to	关于，至于；相对于
be equal to	等于
at hand	在手边，在附近，即将到来
algorithmic complexity	算法复杂性
failure elimination	排错
imperative language	命令语言，强制语言
functional language	函数语言
logic language	逻辑语言
use case	用例
be strong in	擅长
at the price of	以……的代价
be determined by	由……所决定
Java virtual machine	Java 虚拟机
native machine code	本地机器代码

Abbreviations

TSP (Travelling Salesman Problem)	旅行商问题
IDE (Integrated Development Environment)	集成开发环境
COBOL (COmmon Business-Oriented Language)	面向商业的通用语言
PHP (Hypertext Preprocessor)	超文本预处理语言
OOAD (Object-Oriented Analysis and Design)	面向对象分析和设计
MDA (Model-Driven Architecture)	模型驱动架构
UML (Unified Modeling Language)	统一建模语言
ER Modeling (Entity-Relationship Modeling)	实体—关系建模

Notes

[1] This code may be a modification of existing source or something completely new, and the purpose is to create a program that exhibits the desired behavior.

本句中，to create a program 是动词不定式短语，做表语。that support decision-making 是定语从句，修饰和限定 a program。

[2] Another ongoing debate is the extent to which the programming language used in writing programs affects the form that the final program takes.

本句中，which the programming language used in writing programs affects the form that the final program takes 是定语从句，修饰和限定 the extent。在该从句中，used in writing programs 是过去分词短语，做定语，修饰和限定 the programming language；that the final program takes 也是定语从句，修饰和限定 the form。

[3] The discipline differs from many other technical professions in that programmers generally do not need to be licensed or pass any standardized (or governmentally regulated) certification tests in order to call themselves “programmers” or even “software engineers”.

本句中，in that programmers generally do not need to be licensed or pass any standardized (or governmentally regulated) certification tests in order to call themselves “programmers” or even “software engineers”是原因状语从句，修饰主句的谓语 differs from。in that 等于 because。in order to 的意思是“为了”。

[4] For this purpose, algorithms are classified into *orders* using so-called Big O notation, O(n), which expresses execution time, memory consumption, or another parameter in terms of the size of an input.

本句中，using so-called Big O notation, O(n)是现在分词短语，做定语，修饰和限定

orders。which expresses execution time, memory consumption, or another parameter in terms of the size of an input 是非限定性定语从句，对 Big O notation, O(n)进行补充说明。

[5] Research in computer programming includes investigation into the unsolved proposition that *P*, the class of algorithms which can be deterministically solved in polynomial time with respect to an input, is not equal to *NP*, the class of algorithms for which no polynomial-time solutions are known.

本句中，that *P*, the class of algorithms which can be deterministically solved in polynomial time with respect to an input, is not equal to *NP*, the class of algorithms for which no polynomial-time solutions are known 是同位语从句，对 the unsolved proposition 进行解释说明。在该从句中，*P* 是主语，is not equal to *NP* 是谓语；the class of algorithms which can be deterministically solved in polynomial time with respect to an input 是名词性短语，对 *P* 进行补充说明，在该短语中，which can be deterministically solved in polynomial time with respect to an input 是定语从句，修饰和限定 the class of algorithms。for which no polynomial-time solutions are known for which no polynomial-time solutions are known 也是名词性短语，对 *NP* 进行补充说明，在该短语中，for which no polynomial-time solutions are known 是介词前置的定语从句，修饰和限定它前面的 the class of algorithms。

Exercises

【Ex.1】根据课文内容，回答以下问题。

1) What is computer programming?

2) What does the process of writing source code require?

3) What are the general criteria for good programming?

4) What does the programmer’s job involve?

5) What should be considered when choosing the programming language?

6) What should be the first step in every software development project?

7) What do popular modeling techniques include?

8) What do implementation techniques include?

9) What are the methods of measuring language popularity mentioned in the passage?

10) Why is debugging a very important task for every programmer?

【**Ex.2**】英汉互译。

1. Use Case	1.
2. requirements analysis	2.
3. failure elimination	3.
4. native machine code	4.
5. Unified Modeling Language	5.
6. *n*. 符号	6.
7. *vt*. 初始化	7.
8. 面向对象分析和设计	8.
9. *adj*.可维护的	9.
10. 算法复杂性	10.

【**Ex.3**】用下面方框中的词填空。

elements	memory	hiding	language	developing
comparison	amount	portable	details	concepts

In computing, a high-level programming language is a programming language with strong abstraction from the details of the computer. In __1__ to low-level programming languages, it may use natural language __2__ , be easier to use, or be more __3__ across platforms. Such languages hide the __4__ of CPU operations such as __5__ access models and management of scope.

This greater abstraction and __6__ of details is generally intended to make the language user-friendly, as it includes __7__ from the problem domain instead of those of the machine used. A high-level __8__ isolates the execution semantics of a computer architecture from the specification of the program, making the process of __9__ a program simpler and more understandable with respect to a low-level language. The __10__ of abstraction provided defines how “high-level” a programming language is.

【**Ex.4**】把下列短文翻译成中文。

Software Testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. Different software development models will focus the test effort at different points in the development process. In a more traditional model, most of the test effort occurs after the requirements have been defined and the coding process has been completed. Newer development models, such as Agile or XP, often employ test driven development and place an increased portion of the testing up front in the development process, in the hands of the developer.

Text B

C Language—Control Statements

A program consists of a number of statements which are usually executed in sequence. Programs can be much more powerful if we can control the order in which statements are run.

Statements fall into three general types:

- Assignment, where values, usually the results of calculations, are stored in variables.
- Input / Output, data is read in or printed out.
- Control, the program makes a decision about what to do next.

This section will discuss the use of control statements in C. We will show how they can be used to write powerful programs by:

- Repeating important sections of the program.
- Selecting between optional sections of a program.

1. The if else Statement

This is used to decide whether to do something at a special point, or to decide between two courses of action.

The following test decides whether a student has passed an exam with a pass mark of 45:

```
if (result >= 45)
       printf("Pass\n");
else
       printf("Fail\n");
```

It is possible to use the if part without the else.

```
if (temperature < 0)
       print("Frozen\n");
```

Each version consists of a test (this is the bracketed statement following the if). If the test is true then the next statement is obeyed. If it is false then the statement following the else is obeyed if present. After this, the rest of the program continues as normal.

If we wish to have more than one statement following the if or the else, they should be grouped together between curly brackets. Such a grouping is called a compound statement or a block.

```
if (result >= 45)
{      printf("Passed\n");
       printf("Congratulations\n")
}
else
```

```
{       printf("Failed\n");
        printf("Good luck in the resits\n");
}
```

Sometimes we wish to make a multi-way decision based on several conditions. The most general way of doing this is by using the else if variant on the if statement. This works by cascading several comparisons. As soon as one of these gives a true result, the following statement or block is executed, and no further comparisons are performed. In the following example we are awarding grades depending on the exam result.

```
if (result >= 75)
        printf("Passed: Grade A\n");
else if (result >= 60)
        printf("Passed: Grade B\n");
else if (result >= 45)
        printf("Passed: Grade C\n");
else
        printf("Failed\n");
```

In this example, all comparisons test a single variable called result. In other cases, each test may involve a different variable or some combination of tests. The same pattern can be used with more or fewer else if's, and the final lone else may be left out. It is up to the programmer to devise the correct structure for each programming problem.

2. The switch Statement

This is another form of the multiway decision. It is well structured, but can only be used in certain cases where:

- Only one variable is tested, all branches must depend on the value of that variable. The variable must be an integral type (int, long, short or char).
- Each possible value of the variable can control a single branch. A final, catch all, default branch may optionally be used to trap all unspecified cases.

Hopefully an example will clarify things. This is a function which converts an integer into a vague description. It is useful where we are only concerned in measuring a quantity when it is quite small.

```
estimate(number)
int number;
/* Estimate a number as none, one, two, several, many */
{       switch(number) {
        case 0:
                printf("None\n");
                break;
        case 1:
                printf("One\n");
                break;
```

```
        case 2:
                printf("Two\n");
                break;
        case 3:
        case 4:
        case 5:
                printf("Several\n");
                break;
        default:
                printf("Many\n");
                break;
        }
}
```

Each interesting case is listed with a corresponding action. The break statement prevents any further statements from being executed by leaving the switch. Since case 3 and case 4 have no following break, they continue on allowing the same action for several values of number.

Both if and switch constructs allow the programmer to make a selection from a number of possible actions.

The other main type of control statement is the loop. Loops allow a statement, or block of statements, to be repeated. Computers are very good at repeating simple tasks many times, the loop is C's way of achieving this.

3. Loops

C gives you a choice of three types of loop, while, do while and for.

- The while loop keeps repeating an action until an associated test returns false. This is useful where the programmer does not know in advance how many times the loop will be traversed.
- The do while loops is similar, but the test occurs after the loop body is executed. This ensures that the loop body is run at least once.
- The for loop is frequently used, usually where the loop will be traversed a fixed number of times. It is very flexible, and novice programmers should take care not to abuse the power it offers.

3.1 The while Loop

The while loop repeats a statement until the test at the top proves false.

As an example, here is a function to return the length of a string. Remember that the string is represented as an array of characters terminated by a null character '\0'.

```
int string_length(char string[])
{       int i = 0;
        while (string[i] != '\0')
                i++;
        return(i);
}
```

The string is passed to the function as an argument. The size of the array is not specified, the function will work for a string of any size.

The while loop is used to look at the characters in the string one at a time until the null character is found. Then the loop is exited and the index of the null is returned. While the character isn't null, the index is incremented and the test is repeated.

3.2 The do while Loop

This is very similar to the while loop except that the test occurs at the end of the loop body. This guarantees that the loop is executed at least once before continuing. Such a setup is frequently used where data is to be read. The test then verifies the data, and loops back to read again if it was unacceptable.

```
do
{       printf("Enter 1 for yes, 0 for no :");
        scanf("%d", &input_value);
} while (input_value != 1 && input_value != 0)
```

3.3 The for Loop

The for loop works well where the number of iterations of the loop is known before the loop is entered. The head of the loop consists of three parts separated by semicolons.

- The first is run before the loop is entered. This is usually the initialization of the loop variable.
- The second is a test, the loop is exited when this returns false.
- The third is a statement to be run every time the loop body is completed. This is usually an increment of the loop counter.

The example is a function which calculates the average of the numbers stored in an array. The function takes the array and the number of elements as arguments.

```
float average(float array[], int count)
{       float total = 0.0;
        int i;
        for(i = 0; i < count; i++)
                total += array[i];
        return(total / count);
}
```

The for loop ensures that the correct number of array elements are added up before calculating the average.

The three statements at the head of a for loop usually do just one thing each, however any of them can be left blank. A blank first or last statement will mean no initialization or running increment. A blank comparison statement will always be treated as true. This will cause the loop to run indefinitely unless interrupted by some other means. This might be a return or a break statement.

It is also possible to squeeze several statements into the first or third position, separating them with commas. This allows a loop with more than one controlling variable. The example below illustrates the definition of such a loop, with variables hi and lo starting at 100 and 0 respectively and converging.

```
for (hi = 100, lo = 0; hi >= lo; hi--, lo++)
```

The for loop is extremely flexible and allows many types of program behaviour to be specified simply and quickly.

4. The break Statement

We have already met break in the discussion of the switch statement. It is used to exit from a loop or a switch, control passing to the first statement beyond the loop or a switch.

With loops, break can be used to force an early exit from the loop, or to implement a loop with a test to exit in the middle of the loop body. A break within a loop should always be protected within an if statement which provides the test to control the exit condition.

5. The continue Statement

This is similar to break but is encountered less frequently. It only works within loops where its effect is to force an immediate jump to the loop control statement.

- In a while loop, jump to the test statement.
- In a do while loop, jump to the test statement.
- In a for loop, jump to the test, and perform the iteration.

Like a break, continue should be protected by an if statement.

6. The goto Statement

C has a goto statement which permits unstructured jumps to be made. Its use is not recommended, so we'll not teach it here.

New Words

statement	['steitmənt]	*n.* 语句
assignment	[ə'sainmənt]	*n.* 赋值，分配
bracket	['brækit]	*n.* 括号
		v. 括在一起
obey	[ə'bei]	*v.* 服从，顺从
cascade	[kæs'keid]	*n.* 层叠
award	[ə'wɔ:d]	*vt.* 授予，判给
lone	[ləun]	*adj.* 孤独的，独立的
devise	[di'vaiz]	*vt.* 设计，做出（计划），想出（办法）
default	[di'fɔ:lt]	*n.* 默认，缺省

trap	[træp]	*n.* 收集，吸收
unspecified	['ʌn'spesifaid]	*adj.* 未指明的，未详细说明的
clarify	['klærifai]	*v.* 澄清，阐明
prove	[pru:v]	*vt.* 证明，证实，检验
argument	['ɑ:gjumənt]	*n.* 论据，依据，论点
exit	['eksit]	*n.* 出口
		vi. 退出
verify	['verifai]	*vt.* 检验，校验
unacceptable	['ʌnək'septəbl]	*adj.* 不能接受的
semicolon	['semi'kəulən]	*n.* 分号（即";"）
initialization	[i,niʃəlai'zeiʃən]	*n.* 设定初值，初始化
counter	['kauntə]	*n.* 计数器
blank	[blæŋk]	*adj.* 空白的，空着的
squeeze	[skwi:z]	*n.* 压榨，挤
		v. 压榨，挤，挤榨
comma	['kɔmə]	*n.* 逗号
respectively	[ri'spektivli]	*adv.* 分别地，各个地
converging	[kən'və:dʒiŋ]	*adj.* 收敛的；会聚的，趋同的
force	[fɔ:s]	*vt.* 强制，强加
encounter	[in'kauntə]	*v.* 遇到，相遇
unstructured	[ʌn'strʌktʃəd]	*adj.* 非结构的
recommend	[rekə'mend]	*vt.* 推荐，介绍

Phrases

read in …	把……读入
print out	显示，印出
course of action	做法，行动过程；一连串行动
pass mark	及格分数
group together	集合
curly bracket	花括号，波形括号
compound statement	复合语句
be left out	略去
integral type	整形
null character	空字符
one at a time	一次一个
loop body	循环体
add up	合计

at the head of …	在……的最前面
be treated as	被视为
in the middle of …	在……的中间

Exercises

【Ex.5】根据课文内容，回答以下问题。

1) What does a program consist of?

2) What are the three general types of statements?

3) What is the if else statement used to do?

4) What should be done if we wish to have more than one statement following the if or the else?

5) What are the certain cases where the switch statement can only be used?

6) What are the three types of loop given by C?

7) What does the while loop do?

8) What does the for loop ensure?

9) What is the break statement used to do?

10) Where does the continue statement work? How?

Reading Material

Text	Note
Java Language Basics Java is a powerful, cross-platform, object-oriented programming language suitable for writing anything from a	

Text	Note
distributed application that runs on a corporate network to a database-driven Web site to host your personal photo gallery. To make it easier to learn, the Java language was designed to resemble[1] some of the most popular programming languages in use today, most notably C/C++. If you're not a C/C++ expert, however (and most Web developers aren't), the language can be a little intimidating[2]. In this article, I'll bring you up to speed on the basic syntax of the Java language, including variables, data types, operators, and control structures.	[1] *vt.* 像, 类似 [2] *adj.* 令人胆怯的，令人生畏的
1. Variables Here is the listing for the program I helped you create in the previous article:	

`1 /**`
`2 * Hello.java`
`3 * A simple Java program`
`4 */`
`5`
`6 class Hello {`
`7 public static void main(String[] args) {`
`8 // Print a couple of one-line messages`
`9 System.out.println("Hello, World!");`
`10 System.out.println("This is a test.");`
`11 }`
`12 }`	

This is an exceedingly simple program, as Java programs go, and I'm sure you don't need me to tell you that it's quite useless. What good is a program that prints out the same two lines every time you run it? At the very least[3], a program should perform some kind of useful calculation, right?	[3] at the very least: 至少，起码；无论如何
To be able to perform calculations and do useful work, a program must be able to manipulate data in some fashion. Like most programming languages, Java lets you store data in variables. A variable may be thought of simply as a named location in memory where data may be stored. Since different kinds of data may have different storage requirements, Java requires you to specify a data type for every variable that you create.	
Let's look at a basic example to help solidify[4] this concept	[4] *v.* 巩固

Text	Note

in your mind. Say you were writing a program that was performing temperature calculations for a laboratory experiment. Many such experiments take the room temperature into account in their calculations, so your program might need to store the room temperature in a variable. The following code creates a variable for storing whole numbers called roomTemp, and then assigns it a value of 20:
`int roomTemp; // Create integer variable`
`roomTemp = 20;// Assign the variable a value of 20`	

For the Americans in the audience who are used to seeing temperatures in Fahrenheit[5], 20 degrees Celsius[6] is 68 degrees Fahrenheit. int stands for[7] integer (programming lingo[8] for a whole number[9]), and is the data type of the roomTemp variable. So as you can see, creating a variable is as simple as typing the data type followed by the name of the variable. Variable names in Java, like methods, are not capitalized by convention, but it's okay to use uppercase letters within the variable name to make them more readable. This is why I chose to name the variable roomTemp, instead of RoomTemp or roomtemp, or some other variation on the theme.	[5] *adj*. 华氏温度的 *n*. 华氏度 [6] *adj*. 摄氏的 *n*. 摄氏度 [7] stand for: 代表，代替 [8] *n*. 行话 [9] whole number: 整数
The second line assigns a value to the variable that was just created; specifically, it stores the number 20 in the variable. The equals sign (=) is called the assignment operator[10] because it is used to assign values to variables.	[10] assignment operator:赋值运算符
You must always create a variable before assigning it a value, and you'll usually want to assign the variable a value before putting it to use. Trying to assign a value to a variable that does not exist will cause the Java compiler spit out[11] an error message when you try to compile your program. Rather than automatically creating the variable for you, as some other programming languages do, Java errs on the side of caution by assuming that you have mistyped[12] the name of the variable. This helps avoid bugs due to simple typing mistakes.	[11] spit out: 吐出 [12] *vt*. 输错，错误录入，用打字机错打
A newly created variable does not have a value. For example, until we assigned it a value of 20, one might assume that the roomTemp variable had some default value[13], such as zero. In fact, it did not. A variable with no value is said to be null. If you	[13] default value: 默认值

Text	Note

have trouble getting your head around something having no value at all, you could instead think of null as a special value that is assigned to all newly created variables, no matter their data type. Attempting to use a null variable as if it contained a value is one of the most common types of programming mistakes that cause programs to crash. That goes not only for Java programs, but also for programs written in other languages such as C/C++.
A shortcut exists for creating a variable and assigning it a value on the same line. The following code is equivalent to the two lines we saw above:
`int roomTemp = 20;// Create variable and assign value`
Once created and assigned a value, a variable may be used anywhere the value it contains might be used. Let's look at a simple program that prints out the room temperature as stored in a variable. Open your text editor and type the following (remember, don't type the line numbers), then save the file as PrintRoomTemp.java:
`1 /**`
`2 * PrintRoomTemp.java`
`3 * A simple Java program that prints out the temperature`
`4 */`
`5`
`6 class PrintRoomTemp {`
`7 public static void main(String[] args) {`
`8 int roomTemp = 20; // Room temperature`
`9`
`10 // Print out the temperature`
`11 System.out.print("Current room temperature: ");`
`12 System.out.print(roomTemp);`
`13 System.out.print("degrees Celsius\n");`
`14 }`
`15 }`
This program is a lot like the Hello program we saw above, with a few notable exceptions:
Line 8: The first thing this program does is to create a variable called roomTemp and assign it a value of 20.	

Text	Note
Lines 11—13: In Hello, we used System.out.println(...) to print messages on the screen a line at a time. In this program, we have used System.out.print(...) instead. The only difference between these two commands is that println prints a line break at the end of its output, so that the next character to be displayed on the screen will appear at the start of the next line. Print doesn't add this line break, so several print commands may be strung together to print a single line on the screen.	
Line 12: This is an example of using a variable where a value would normally be expected. Everywhere else that we've used print or println, we've fed it a string of text (surrounded[14] by double quotes) as the exact value[15] to be printed out. In this case, we've given it a variable name instead. For the print command to know what to print out, it has to look inside the variable for the value stored within. So this line just prints out whatever value is stored in the roomTemp variable at the time[16].	[14] *vt.* 包围 [15] exact value: 准确值 [16] at the time: 当时
Line 13: Since this print statement represents the end of the line of text that we want to display, you could use a println instead to output the requisite line break. Instead, I elected to stick with a print statement to demonstrate how to output a line break without using println. Notice that the text string to be printed out by this line ends with \n. This is a special character code that, when printed out by Java, gets converted to a line break[17]. If a situation arises where you want to actually print out a backslash[18] followed by the letter "n", you must prefix[19] your backslash with a second backslash (i.e. "\\n"). The first backslash cancels the special meaning of the second. In other words, \\ is a special character code that gets converted to a single backslash when printed out, so Java no longer sees the \n as a special character code.	[17] line break: 换行符 [18] *n.* 反斜线符号 [19] *n.* 前缀
Compile the program as usual by typing javac PrintRoomTemp.java at the command line, and then, assuming the compiler didn't point out any typing errors, run the compiled version of the program: `D:\java\JavaLanguageBasics>java PrintRoomTemp Current room temperature: 20 degrees Celsius` Although there were a few new tricks in this program, its	

Text	Note
main purpose was to demonstrate how to use a value stored in a variable. In this case, we printed the value out, but as you'll see in later examples, variables can be used in many other ways as well.	
2. Data Types	
In the previous example, we identified the roomTemp variable as being of the int data type, meaning that it could store integer (whole number) values. Now let's say you keep your laboratory a little warmer than most, so you wanted to record a temperature of 22.5 degrees. You might modify line 8 of PrintRoomTemp.java so that it read as follows:	
`int roomTemp = 22.5; // Room temperature`	
If you tried to compile this modified program, however, you would see an error message much like this one:	

`D:\java\JavaLanguageBasics>javac PrintRoomTemp. java`
`PrintRoomTemp.java:8: possible loss of precision found : double`
`required: int`
`int roomTemp = 22.5; // Room temperature`
 	
`1 error`	

Java is telling you that the int data type can't store the value 22.5, and that you should instead use the double type, which is the Java data type for storing floating point numbers (numbers requiring a decimal point[20]). The message "possible loss of precision[21]" refers to the fact that the int data type could store 22 or 23, but not 22.5; that is, int does not provide the level of precision required to represent the value you're trying to store.	[20] decimal point: 小数点 [21] *n.* 精确
So to allow for precise temperature values, you need to declare[22] the roomTemp variable as a double instead of as an int:	[22] *vt.* 声明
`double roomTemp = 22.5; // Room temperature`	
Try this corrected line in your program, and you should see the expected output.	
Java provides a handful of[23] data types for use in your programs. Here's the complete list for the technically minded:	[23] a handful of: 一把
boolean: true or false char: a single 16-bit Unicode character[24]	[24] Unicode character: 统一码

Text	Note
byte: an 8-bit signed integer[25] short: a 16-bit signed integer int: a 32-bit signed integer long: a 64-bit signed integer float: a 32-bit floating-point number double: a 64-bit floating point number In case you're wondering what the number of bits means for these various data types, basically the more bits a data type uses, the more different values it can store. Thus, long variables can store larger numbers than int variables, and double variables can store more precise values than float variables. In most programs you'll be able to get by using int, double, and boolean.	[25] signed integer: 带符号整数

参考译文

计算机编程

计算机编程（通常缩写为编程或编码）是编写、测试及维护计算机程序源代码的过程。源代码用计算机编程语言编写。这个代码可以是原有代码的修改，也可以是全新编写的，目的是建立一个可以实现期望行为的程序。编写源代码的过程需要许多领域的专业技术，包括应用领域、特定的算法以及形式逻辑的知识。

在软件工程中，编程（实现）被认为是软件开发过程中的一个阶段。

在某些特殊的应用或极端情况下，可以这样编写或修改（打补丁）一个程序：直接把要执行的机器代码指令的数值存储到内存中。

现在正进行一个广泛的争论是，编写计算机程序是艺术、手艺还是工程学。好的程序通常考虑满足以下三个方面：一流的专业知识、高效以及可维护的软件解决方案（“高效”和“可维护”的标准差异很大）。这个学科与其他许多技术行业不同，程序员通常无须执照或者通过任何标准的（或政府管制的）认证考试，就可以称为“程序员”或者“软件工程师”。

目前另一个广泛的争论是，编写程序的编程语言对最终程序的形成到底有多大的影响。这类似于语言学中的萨皮尔—沃尔夫假说。

1. 程序员

计算机程序员就是编写计算机软件的人。他们的工作通常包括：需求分析、开发说明、软件体系设计、编写代码、编译、软件测试、编写文档、系统集成和软件维护。

2. 编程语言

不同的编程语言支持不同的编程风格（叫做编程范式）。选择所用语言应考虑许多因素，如公司政策、对任务的适应性、第三方软件包的有效性以及个人偏爱。最理想的是选择最适合完成手头任务的编程语言。主要权衡以下方面：是否可以找到足够了解所用语言的程序员建立团队、该语言编译器的有效性以及用该语言编写的程序的执行效率。

3. 现代编程

3.1 算法的复杂性

在计算机编程的学术领域和工程实践中，主要关注的是对给定的问题找出最有效的算法并实现它。为此，算法使用大 O 记法分类，O(n)表示执行的时间和内存消耗，另一个参数是输入的规模。有经验的程序员熟悉多种广泛认可的算法及其复杂性，并据此考虑设计方案，例如权衡内存消耗和性能。

计算机编程的研究包括寻找未解决的命题：P 不等于 NP。P 这类算法根据输入的多项式时间可以确定性地解决。NP 没有多项式时间解决方案。研究表明，许多 NP 算法可以在多项式时间内转换为其他算法，如旅行商问题，这样可以建立一大类“难题”，以便分析。

3.2 方法学

每个软件项目开发的第一步都应该是需求分析，然后是建模、执行和排错（调试）。

完成这些任务中的任意一项都会有许多不同的方法。常用的需求分析方法是用例分析。

常用的建模技术包括面向对象分析和设计及模型驱动架构。统一建模语言是这两者都用的表示法。

数据库设计中使用的类似技术是实体—关系建模（ER 建模）。

执行技术包括命令语言（面向对象或面向过程）、函数语言及逻辑语言。

通常使用像 Visual Studio 和 Eclipse 这样的 IDE 进行调试。也使用像 gdb 的独立调试器。

3.3 衡量语言的效用

确定现代编程语言中最流行的语言是很困难的。对于某种特定的应用某些语言是非常流行的（例如，Cobol 在企业数据中仍然坚挺，常用于大型计算机；Fortran 用于工程，而 C 用于嵌入式应用系统），而一些语言可以编写多种应用程序。

衡量语言流行程度的方法包括计算提及该语言的招聘广告数，讲述该语言图书的销售量（对新语言这可能会有所夸大），估计已经用该语言编写的程序行数（可能会低估像 Cobol 这样的商业语言的用户数）。

3.4 调试

对每个程序员来说，调试是非常重要的任务，因为错误的程序往往无用。像 C++和汇编语言，即使对专业人员也很有挑战性，因为要调试像缓冲器溢出、错误指针或未初始化内存这样的错误是很困难的。缓冲器溢出可能损害附近的内存区域，并引起一个完全不同

的程序行的失效。因此，像 Valgrind、Purify 或 Boundschecker 这样的内存分配工具对用 C++语言进行现代软件开发是必须的。像 Java、PHP 和 Python 这样的语言可以防止程序员犯大多数运行期错误，但可能其代价是极大地降低，最终程序的运行速度。在执行速度由其他因素（如数据库访问或文件 I/O 处理）决定时，这是可以接受的。准确的代价取决于特定的执行细节。例如，现代的 Java 虚拟机使用了多种高级优化技术，包括在运行期把解释指令转换为本地机器码。