1 读取输入

要想通过控制台进行输入，首先需要构造一个与标准输入流`System.in`关联的`Scanner`对象：

```
Scanner in = new Scanner(System.in);
```

之后就可以使用`Scanner`类的各种方法读取输入了。下面列出`Scanner`类中关于读取输入的方法。

```
/* java.util.Scanner */
Scanner(InputStream in) // 用给定的输入流创建一个 Scanner 对象
String nextLine() // 读取输入的下一行内容
String next() // 读取输入的下一个单词，以空格作为分隔符
int nextInt() // 读取并转换下一个表示整数的字符序列
double nextDouble() // 读取并转换下一个表示浮点数的字符序列
boolean hasNext() // 检测输入中是否还有其他单词
boolean hasNextInt() // 检测是否还有下一个表示整数的字符序列
boolean hasNextDouble() // 检测是否还有下一个表示浮点数的字符序列
```

`Scanner`类定义在`java.util`包中，要使用`import`命令导入相应的包。程序实例：

```
import java.util.*;

pubilc class InputTest
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        System.out.print("What is your name? ");
        String name = in.nextLine();

        System.out.print("How old are you? ");
        int age = in.nextInt();

        System.out.println("Hello, " + name + ". Next year, you'll be " + (age + 1));
    }
}
```

因为输入是可见的，所以`Scanner`类不适用于从控制台读取密码。Java 6 特别引入了`Console`类来实现密码输入。下面列出与此相关的一些方法。

```
/* java.lang.System */
static Console console()
    // 如果可以进行交互，就返回一个 Console 对象通过控制台窗口与用户交互，否则返回 null
    // 对于任何一个在控制台窗口启动的程序，都可使用 Console 对象。否则，是否可用取决于所使用的系统

/* java.io.Console */
static char[] readPassword(String prompt, Object... args)
static String readLine(String prompt, Object... args)
    // 显示字符串 prompt（提示符）并读取用户输入，直到输入行结束。args 参数可以用来提供格式参数
```

要想读取一个密码，可以使用下列代码：

```
Console cons = System.console();
String username = cons.readLine("User name: ");
char[] passwd = cons.readPassword("Password: ");
```

为安全起见，返回的密码存放在一个字符数组中。在对密码处理完成之后，应该马上用一个填充值覆盖数组元素。

采用`Console`对象处理输入不如采用`Scanner`方便。必须每次读取一行输入，而没有能够读取单个单词或数值的方法。

2 格式化输出

Java 5 沿用了 C 语言函数库中的`printf`方法实现格式化输出。与 C 语言类似，每个以`%`字符开始的格式说明符都用相应的参数替换，格式说明符尾部的转换符指示要格式化的数值的类型。下表列出了所有转换符。（日期时间转换符`Tx`已经过时，故下表未列出）

转换符	类型	示例
`d`	十进制整数	159
`x`	十六进制整数	`9f`
`o`	八进制整数	237
`f`	定点浮点数，默认 6 位小数	15.9
`e`	指数浮点数	`1.59e+01`

转换符	类型	示例
g	通用浮点数（e和f中较短的一个）	——
a	十六进制浮点数	0x1.fccdp3
s	字符串	Hello
c	字符	H
b	布尔	true
h	散列码	42628b2
%	百分号	%
n	与平台有关的行分隔符	——

另外，还可以在%和转换符之间指定控制格式化输出外观的各种标志。下表列出了所有标志。

标志	目的	示例
n.m	n为字段宽度，m为小数点后位数。默认右对齐。 若长度不足，默认用空格补齐	3333.33
+	打印正数和负数的符号	+3333.33
空格	正数前面添加空格	/ 3333.33/
0	若长度不足，数字前面补 0	03333.33
-	左对齐。不能与0连用	/3333.33 /
(	将负数括在括号内（不包括负号）	(3333.33)
,	添加分组分隔符	3,333.33
# （对于f格式）	当小数点被截去时，包含小数点	3333.
#(对于x或o格式)	添加前缀0x或0	0xcafe
$	指定要格式化的参数索引	——
<	格式化前面说明的数值	——

用`n.m`控制字段宽度和精度时，若实际长度大于 n，则无视该宽度限制。`+`、`-`、`(`、`)`、`.`等符号也计入长度。若小数点后位数多于 m，则截取前 m 位；若小数点后位数少于 m，则用 0 补齐。若 m 为 0，则不输出小数点和小数部分，此时若使用`#`标志，仍输出小数点而小数部分不输出。

`$`标志用于指定要格式化的参数索引。参数索引紧跟在`%`后面，并以`$`终止。参数索引从 1 开始。例如：

```
System.out.printlf("%1$f %1$.2f %2$d", -3333.3333, 21);
// 输出: -3333.333300 -3333.33 21
```

`<`标志指示前一个格式说明中的参数将被再次使用。例如：

```
System.out.printf("%f %<.2f", -3333.3333);
// 输出: -3333.333300 -3333.33
```

与格式化输出类似，可以使用`String`类的静态方法`format`创建一个格式化的字符串。例如：

```
String message = String.format("Hello, %s. Next year, you'll be %d", name, age);
```

3 文件输入与输出

文件读写涉及的 API 如下：

```
/* java.util.Scanner */
Scanner(Path p, String encoding) // 构造一个使用给定字符编码从给定路径读取数据的 Scanner

/* java.io.PrintWriter */
PrintWriter(String fileName) // 构造一个将数据写入文件的 PrintWriter，文件名由参数指定

/* java.nio.file.Path */
static Path of(String pathname) // 根据给定的路径名构造一个 Path 对象
```

3.1 读取文件

先构造一个`Scanner`对象：

```
Scanner in = new Scanner(Path.of("myfile.txt"), StandardCharsets.UTF_8);
```

之后就可以使用前面介绍的任何一个`Scanner`方法来读文件。

注意：构造`Scanner`对象时第二个参数表示字符编码，不可省略。

3.2 写入文件

构造一个`PrintWriter`对象：

```
PrintWriter out = new PrintWriter("myfile.txt");
```

如果文件不存在，创建该文件。之后就可以使用`print`、`println`、`printf`等方法进行输出。

如果用一个不存在的文件构造`Scanner`，或者用一个无法创建的文件名构造`PrintWriter`，会产生异常。需要在`main`方法中使用`throws`子句标记，如下所示：

```
public static void main(String[] args) throws IOException // 输入/输出异常
{
    Scanner in = new PrintWriter("myfile.txt", StandardCharsets.UTF_8);
    // ...
}
```