《Java 语言程序设计（MOOC 版）》-例题

清华大学出版社　阚道宏主编

第 1 章　认识 Java 语言

例 1-1　一个用 C 语言编写的温度换算程序

```
1  /*
2     一个 C 程序实例：
3     将摄氏温度换算成华氏温度。
4  */
5  #include <stdio.h>    //  插入头文件 stdio.h
6
7  int main( )    //  主函数
8  {
9        double ctemp, ftemp;          //  定义保存温度数据的变量
10       scanf( “%lf”, &ctemp );       //  输入摄氏温度
11       ftemp = ctemp *1.8 +32;    //  计算华氏温度
12       printf( “%lf\n”, ftemp );     //  输出华氏温度
13       return 0;
14 }
```

例 1-2　一个用 C++语言编写的温度换算程序

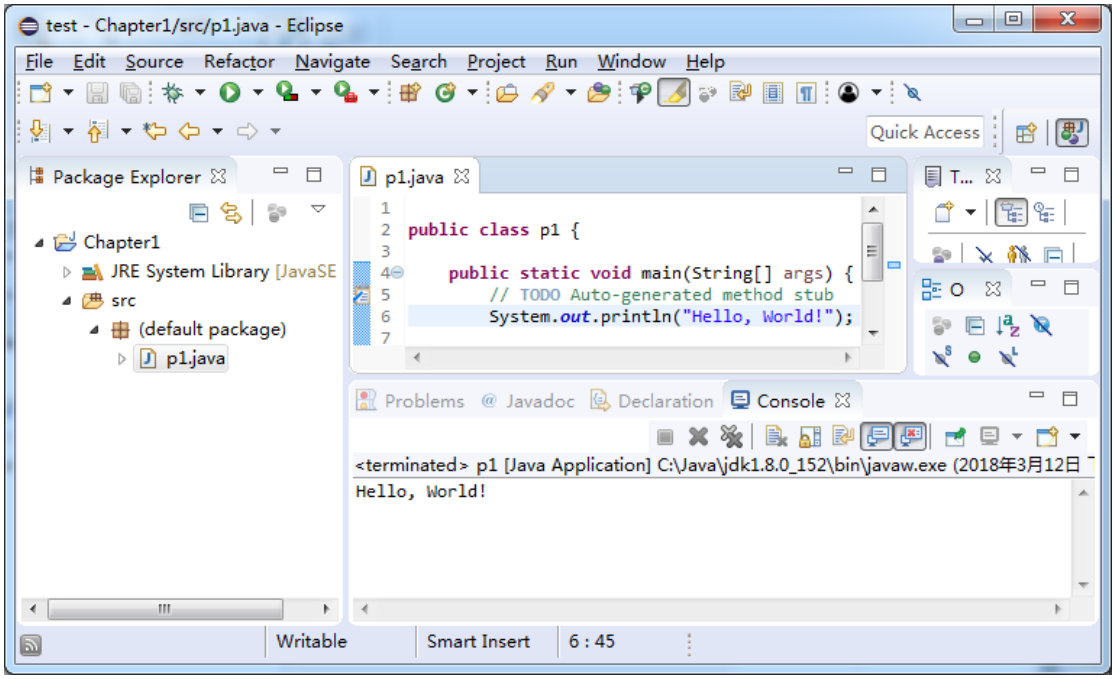
```
1  /*
2     一个 C++程序实例：
3     将摄氏温度换算成华氏温度。
4  */
5  #include <iostream>       //  插入头文件 iostream
6  using namespace std;     //  声明命名空间 std
7
8  int main( )    //  主函数
9  {
10       double ctemp, ftemp;       //  定义保存温度数据的变量
11       cin >> ctemp;                  //  输入摄氏温度
12       ftemp = ctemp *1.8 +32;    //  计算华氏温度
13       cout << ftemp;                 //  输出华氏温度
14       return 0;
15 }
```

例 1-3 一个用 Java 语言编写的温度换算程序

```
1  /*
2     一个 Java 程序实例：
3     将摄氏温度换算成华氏温度。
4  */
5  import java.util.Scanner;   // 导入外部程序 Scanner
6
7  public class JavaTemp {                                  // 先定义一个类
8        public static void main( String args[ ] )  {      // 将主函数定义在类的里面
9              double ctemp, ftemp;                        // 定义保存温度数据的变量
10             Scanner sc = new Scanner( System.in );   // 创建键盘扫描器对象
11             ctemp = sc.nextDouble();                    // 输入摄氏温度
12             ftemp = ctemp * 1.8 + 32;                   // 计算华氏温度
13             System.out.println( ftemp );                // 输出华氏温度
14             return;
15       }
16 }
```

例 1-4 简单 Java 程序的代码框架

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class 类名 {   // 先定义一个类，类名需与源程序文件名一致
4                        // 假设类名为“P1”，则源程序文件名必须为“P1.java”
5        public static void main( String args[ ] )  {      // 将主函数定义在类的里面
6              int x;   double y;                          // 定义变量，申请内存
7              Scanner sc = new Scanner( System.in );   // 创建键盘扫描器对象
8              x = sc.nextInt();   y = sc.nextDouble();   // 使用扫描器输入原始数据
9              ……;                                        // 编写表达式进行数据处理
10             System.out.println( …… );                  // 输出计算结果
11       }
12 }
```



第 2 章 Java 语言基础

例 2-5 实现求倒数算法的 Java 程序（if-else 语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                              // 主类
4      public static void main(String[] args) {         // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          double x;                // 定义一个 double 型变量 x
7          x = sc.nextDouble();   // 键盘输入变量 x 的值
8
9          if (x != 0) {   // 判断条件“x 不等于 0”是否成立
10             // 条件成立时执行下列代码。因为是多条语句，所以用{ }括起来
11             double y;   // 再定义一个 double 型变量 y，用于保存 x 的倒数
12             y = 1 / x;   // 求 x 的倒数，结果赋值给 y
13             System.out.println( y );   // 显示 y 的值，即 x 的倒数
14         }
15         else   // else 分支只有一条语句，可省略大括号
16             System.out.println( "0 的倒数没有意义" );   // 显示错误信息
17     }
18 }
```

例 2-6 判断年份是否闰年的 Java 程序

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                              // 主类
4      public static void main(String[] args) {         // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int year;                 // 定义一个 int 型变量 year
7          year = sc.nextInt();   // 键盘输入一个年份，保存到变量 year 中
8
9          if ( (year%4 == 0 && year%100 != 0) || year%400 == 0 )   // 判断闰年条件是否成立
10             System.out.println( year + "是闰年" );      // 条件成立则该年份是闰年
11         else
12             System.out.println( year + "不是闰年" );   // 否则该年份不是闰年
13     }
14 }
```

例 2-7 求符号函数 sgn(x)的 Java 程序

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                              // 主类
4      public static void main(String[] args) {         // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          float x;                  // 定义一个 float 型变量 x
7          x = sc.nextFloat();   // 键盘输入变量 x 的值
8
9          int sgn;      // 定义一个 int 型变量 sgn，用于保存符号函数的结果
10         if (x == 0)      // 首先将 x 分为等于 0 和不等于 0 两种情况
11             sgn = 0;   // x = 0 的情况
12         else {             // 在 x 不等于 0 时，再进一步区分 x>0 和 x<0 这两种情况
13             if (x > 0)    sgn = 1;   // x > 0 的情况
14             else    sgn = -1;        // x < 0 的情况
15         }
16         System.out.println( sgn );   // 显示 sgn 的值，即符号函数的结果
17     }
18 }
```

例 2-8 求符号函数 sgn(x)的 Java 程序（if-else if 语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          float x;                  // 定义一个 float 型变量 x
7          x = sc.nextFloat();   // 键盘输入变量 x 的值
8
9          int sgn;   // 定义一个 int 型变量 sgn，用于保存符号函数的结果
10         if (x == 0)    sgn = 0;        // 首先检查 x 等于 0 的情况
11         else if (x > 0)    sgn = 1;   // 再检查 x 大于 0 的情况
12         else    sgn = -1;             // 最后剩下的就是 x 小于 0 的情况
13         System.out.println( sgn );   // 显示符号函数的结果
14     }
15 }
```

例 2-9 显示星期几英文单词的 Java 程序

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int x;                  // 定义一个 int 型变量 x
7          x = sc.nextInt();   // 键盘输入一个表示星期几的数值（1~7），保存到变量 x 中
8
9          // 下列 if-else if 语句根据 x 的值显示其对应的英文单词
10         if (x == 1)    System.out.println( "Monday" );
11         else if (x == 2)    System.out.println( "Tuesday" );
12         else if (x == 3)    System.out.println( "Wednesday" );
13         else if (x == 4)    System.out.println( "Thursday" );
14         else if (x == 5)    System.out.println( "Friday" );
15         else if (x == 6)    System.out.println( "Saturday" );
16         else if (x == 7)    System.out.println( "Sunday" );
17         else    System.out.println( "Input Error" );   // 输入数值不在 1~7 范围之内，提示错误
18     }
19 }
```

例 2-11 显示星期几英文单词的 Java 程序（switch-case 语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int x;                  // 定义一个 int 型变量 x
7          x = sc.nextInt();   // 键盘输入一个表示星期几的数值（1~7），保存到变量 x 中
8          // 下列 switch-case 语句根据 x 的值显示对应的英文单词
9          switch ( x ) {
10         case 1:   System.out.println( “Monday” );    break;
11         case 2:   System.out.println( “Tuesday” );    break;
12         case 3:   System.out.println( “Wednesday” );    break;
13         case 4:   System.out.println( “Thursday” );    break;
14         case 5:   System.out.println( “Friday” );    break;
15         case 6:   System.out.println( “Saturday” );    break;
16         case 7:   System.out.println( “Sunday” );    break;
17         default:   System.out.println( “Input Error” );    break;
18         }
19         // 每个 case 语句显示出对应的英文单词之后，程序功能即已完成
20         // 因此使用 break 语句中途跳出 switch 语句
21 }  }
```

例 2-12 显示不同月份天数的 Java 程序（switch-case 语句：共用语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int month;                  // 定义一个 int 型变量 month
7          month = sc.nextInt();   // 键盘输入一个月份（1~12），保存到变量 month 中
8          // 下列 switch-case 语句显示不同月份的天数
9          switch ( month ) {
10         case 1:    // 1 月大
11         case 3:    // 3 月大
12         case 5:    // 5 月大
13         case 7:    // 7 月大
14         case 8:    // 8 月大
15         case 10:   // 10 月大
16         case 12:   System.out.println( “31 天” );   break;   // 1、3、5、7、8、10、12 月共用语句
17         case 4:    // 4 月小
18         case 6:    // 6 月小
19         case 9:    // 9 月小
20         case 11:   System.out.println( “30 天” );          break;   // 4、6、9、11 月共用语句
21         case 2:    System.out.println( “28 或 29 天” );   break;   // 2 月
22         default:   System.out.println( “Input Error” );    break;   // 提示错误信息
23         }
24 }  }
```

例 2-14 求解奇数数列前 N 项累加和的 Java 程序（while 语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int N;                // 定义一个 int 型变量 N
7          N = sc.nextInt();   // 键盘输入变量 N 的值
8
9          int n = 1, sum = 0;   // 定义循环变量 n（初始值为 1），
10         // 定义保存累加结果的变量 sum（初始值为 0）
11         while (n <= N) {          // 用小括号将循环条件 n<=N 括起来
12             sum += 2*n - 1;   // 将当前项的值 2n-1 累加到 sum 上
13             n++;   // 将 n 加 1，准备下一次累加。该语句使得循环条件 n<=N 趋向于 false
14             // 执行完循环体最后一条语句之后，转到第 11 行，重新判断循环条件
15         }
16         // 如果循环条件不成立，则循环结束，继续执行 while 语句的下一条语句
17         System.out.println( sum );   // 显示变量 sum 的值，即前 N 项的累加和
18     }
19 }
```

例 2-15 求解奇数数列前 N 项累加和的 Java 程序（do-while 语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int N;                // 定义一个 int 型变量 N
7          N = sc.nextInt();   // 键盘输入变量 N 的值
8
9          int n = 1, sum = 0;   // 定义循环变量 n（初始值为 1），
10         // 定义保存累加结果的变量 sum（初始值为 0）
11         do {                      // 先执行循环体
12             sum += 2*n - 1;   // 将当前项的值 2n-1 累加到 sum 上
13             n++;                  // 将 n 加 1，准备下一次累加
14         } while (n <= N) ;        // 后判断条件。如条件成立则重复执行循环体，否则结束循环
15         // 循环结束后，继续执行 do-while 语句的下一条语句
16         System.out.println( sum );   // 显示变量 sum 的值，即前 N 项的累加和
17     }
18 }
```

例 2-16 求解奇数数列前 N 项累加和的 Java 程序（for 语句）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          int N;                  // 定义一个 int 型变量 N
7          N = sc.nextInt();   // 键盘输入变量 N 的值
8
9          int n, sum = 0;   // 定义循环变量 n
10                            // 定义保存累加结果的变量 sum（初始值为 0）
11         for (n = 1; n <= N; n++) {   // for 语句集中用 3 个表达式指定 n 的初始值 1、循环条件 n<=N
12                                       // 以及修改循环变量 n++，使循环条件趋向于 false
13             sum += 2*n - 1;      // 循环体被简化了，原来的 n++语句被放入到 for 语句里面
14         }  // 循环体只有一条语句，此时这对大括号可以省略
15         System.out.println( sum );   // 显示变量 sum 的值，即前 N 项的累加和
16     }
17 }
```

例 2-17 一个计算圆面积的 Java 程序

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          double r;   // 定义一个变量 r 来存放圆的半径
7
8          r = sc.nextDouble();                  // 键盘输入圆的半径
9          System.out.println( 3.14*r*r );   // 显示圆面积
10     }
11 }
```

例 2-18 一个计算圆面积的 Java 程序（break 语句应用示例）

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class JavaTest {                                  // 主类
4      public static void main(String[] args) {             // 主方法
5          Scanner sc = new Scanner( System.in );   // 创建扫描器对象 sc
6          double r;   // 定义一个变量 r 来存放圆的半径
7
8          while ( true ) {                // 死循环
9              r = sc.nextDouble();   // 键盘输入圆的半径
10             if (r <= 0)    break;     // 如果用户输入的半径小于或等于 0，则跳出循环
```

```
11                  System.out.println( 3.14*r*r );   // 显示圆面积
12              }
13              // 使用 break 语句中途跳出 while 语句，继续执行 while 语句的下一条语句
14          }
15  }
```

例 2-19 生成乘法表的 Java 程序 （多重循环应用示例）

```
1   public class JavaTest {   // 主类
2
3       public static void main(String[] args) {   // 主方法
4           int x, y;                           // 定义两个循环变量 x 和 y
5           for (x = 1; x <= 9; x++)   {     // 第一重循环，x 从 1 到 9，共 9 行
6               for (y = 1; y <= x; y++)   // 第二重循环，y 从 1 到 x。第 x 行有 x 个乘法
7                   System.out.print( y +"×" +x +"=" +(x*y) +" " );
8               System.out.print( '\n' );   // 换一行，再显示后续的内容
9           }
10  }   }
```

例 2-20 显示 1~50 之间所有能被 3 整除的数（continue 语句应用示例）

```
1   public class JavaTest {   // 主类
2
3       public static void main(String[] args) {   // 主方法
4           for (int n = 1; n <= 50; n++)   {       // 从 1 到 50 的循环
5               if (n%3 != 0)    continue;        // 如果 n 不能被 3 整除，则执行 continue 语句
6               // continue 语句的作用是结束本次循环，中途返回，去检查下一个数
7               // 未中途返回的数是能被 3 整除的数，下面将显示这些数并用逗号隔开
8               System.out.print( n +", " );
9           }
10  }   }
```

例 2-21 显示 100~200 之间的所有质数（带标号的 continue 语句应用示例）

```
1  public class JavaTest {                          // 主类
2      public static void main(String[] args) {    // 主方法
3          int i, j, n = 0;
4
5          Loop1: for (i = 100; i <= 200; i += 2) {   // 外层循环，语句块标号 Loop1
6              Loop2: for(j = 2; j <= i/2; j++) {     // 内层循环，语句块标号 Loop2
7                  if ( i%j == 0 )                    // 不是质数，则中途返回
8                      continueLoop1;                 // 借助标号 Loop1，直接返回外层循环
9              }
10             System.out.print( " " +i );   // 是质数：显示质数，以空格隔开
11             n++;                          // 统计显示的指数个数，一行显示 10 个
12             if ( n < 10 )        // 未满 10 个，则不换行
13                 continue;   // 中途返回。无标号时直接返回本层循环，此处也为外层循环
14             System.out.println( );   n = 0;   // 换行显示，并将计数清零
15         }
16     }
17 }
```

第 3 章 面向对象程序设计之一

例 3-1 计算长方形面积和周长的 C++程序代码（**函数**）

程序员甲：主函数（1.cpp）

```
1  #include <iostream>     // C++语言的头文件
2  using namespace std;   // 声明命名空间
3  // C 语言：#include <stdio.h>
4  #include "2.h"   // 插入头文件 2.h
5
6  int main( )
7  {
8      int a, b;   // 定义保存长宽数据的变量
9      cin >> a >> b;     // 输入长方形的长宽
10     // C 语言：scanf("%d %d", &a, &b);
11
12     cout <<Area(a, b) << endl;   // 长方形面积
13     cout <<Len(a, b) << endl;     // 长方形周长
14     // C 语言：printf("%d\n", Area(a, b));
15     // C 语言：printf("%d\n", Len(a, b));
16     return 0;
17 }
```

程序员乙：子函数（2.cpp）

```
// 计算长方形面积和周长的函数
int Area(int length, int width)
{
    return ( length*width );
}
int Len(int length, int width)
{
    return ( 2*(length+width) );
}
```

程序员乙：头文件（2.h）

```
// 声明外部函数的原型
int Area(int length, int width);
int Len(int length, int width);
```

例 3-2 计算长方形面积和周长的 C++程序代码（**结构体类型**）

程序员甲：主函数（1.cpp）

```
1  #include <iostream>
2  using namespace std;
3  #include "2.h"   // 插入头文件 2.h
4
5  int main( )
6  {
7      // ~~int a, b;~~   // 删除该定义变量语句
8      // 改用结构体类型 Rectangle 定义变量
9      struct Rectangle   rect;   // 定义结构体变量
10
11     cin >> rect.a >> rect.b;
12     // 用 rect 的下属成员 a 保存长度
13     // 用 rect 的下属成员 b 保存宽度
14
15     // 调用子函数求长方形的面积和周长
16     cout << Area( rect.a, rect.b ) << endl;
17     cout << Len( rect.a, rect.b ) << endl;
18     return 0;
19 }
```

程序员乙：子函数（2.cpp）

```
// 计算长方形面积和周长的函数
int Area(int length, int width)
{   return ( length*width );   }
int Len(int length, int width)
{   return ( 2*(length+width) );   }
```

程序员乙：头文件（2.h）

```
// 定义一个长方形结构体类型
struct Rectangle
{
    int a;   // 保存长度的成员 a
    int b;   // 保存宽度的成员 b
};
// 声明外部函数的原型
int Area(int length, int width);
int Len(int length, int width);
```

例 3-3 计算长方形面积和周长的 C++程序代码（**类与对象**）

程序员甲：主函数（1.cpp）

```
1  #include <iostream>
2  using namespace std;
3  #include "2.h"   // 插入头文件 2.h
4
5  int main( )
6  {
7      // ~~struct Rectangle   rect;~~   // 删除该语句
8      // 改用类 Rectangle 定义变量（即对象）
9      Rectangle   rect;   // 定义一个长方形对象 rect
10
11     cin >> rect.a >> rect.b;
12     // 用 rect 的数据成员 a 保存长度
13     // 用 rect 的数据成员 b 保存宽度
14
15     // 调用 rect 的函数成员求其面积和周长
16     cout << rect.Area( ) << endl;//不需要传递长宽
17     cout << rect.Len( ) << endl;
18     return 0;
19 }
```

程序员乙：类实现程序文件（2.cpp）

```
#include “2.h”   // 插入头文件 2.h
// 定义长方形类 Rectangle：类实现部分
// 给出各函数成员的完整定义代码
int Rectangle::Area( )//不需要传递长宽
{   return ( a*b );   }
int Rectangle::Len( )
{   return ( 2*(a+b) );   }
```

程序员乙：类声明头文件（2.h）

```
// 定义一个长方形类 Rectangle
// 定义类的代码分为声明和实现两部分
class Rectangle   // 类声明部分
{
public:
    int a;   // 数据成员：保存长度
    int b;   // 数据成员：保存宽度
    int Area( );   // 函数成员：计算面积
    int Len( );    // 函数成员：计算周长
};
// 类 Rectangle 的实现部分放在 2.cpp 文件中
```

例 3-5 计算长方形面积和周长的 Java 程序代码（**类与对象**）

程序员甲：主类文件（RectangleTest.java）

```
1  public class RectangleTest {   // 主类
2
3      // 将主函数 main()定义在类中
4      public static void main(String[] args) {
5          // Java 需要动态创建对象
6          Rectangle obj = new Rectangle();
7
8          obj.Input();     // 输入长宽
9          obj.Output();   // 显示结果
10     }
11
12 }
13
14
15
16
17
18
19
```

程序员乙：长方形类文件（Rectangle.java）

```
import java.util.Scanner;   // 导入外部程序 Scanner

public class Rectangle {     // 长方形类定义代码
    private double a, b;   // 字段：保存长度和宽度
    private double Area( )   // 方法：计算面积
    {   return a*b;   }
    private double Len( )     // 方法：计算周长
    {   return 2*(a+b);   }

    public void Input( ) {     // 方法：输入长宽
        // 创建键盘扫描器对象
        Scanner sc = new Scanner( System.in );
        // 然后通过键盘扫描器对象输入长宽
        a = sc.nextDouble();   b = sc.nextDouble();
    }
    public void Output( ) {   // 方法：输出结果
        System.out.println( Area() +", " +Len() );
    }
}
```

例 3-6 将主方法 main()定义在长方形类 Rectangle 中

```
1  import java.util.Scanner;   // 导入外部程序 Scanner
2
3  public class Rectangle {     // 长方形类定义代码
4      private double a, b;   // 字段：保存长度和宽度
5      private double Area( )  {  ……  }  // 代码省略
6      private double Len( )    {  ……  }  // 代码省略
7      public void Input( )      {  ……  }  // 代码省略
8      public void Output( )    {  ……  }  // 代码省略
9
10     // 将主方法 main()定义在长方形 Rectangle 类中
11     public static void main(String[] args) {
12         Rectangle obj = new Rectangle();
13         obj.Input();       // 输入长宽
14         obj.Output();   // 显示结果
15     }
16     // 语法上，主方法也是类的一个成员，放在其他类成员的前面或后面都可以
17 }
```

例 3-7 测算养鱼池工程总造价的 Java 程序代码（面向对象程序设计方法）

程序员甲：主类+主方法（Pool.Java）

```
1   import java.util.Scanner;   // 导入外部程序 Scanner
2
3   public class Pool {                          // 主类
4     public static void main(String[] args) {   // 主方法
5       Scanner sc = new Scanner( System.in );
6       double totalCost = 0;   // 保存总造价的变量
7       // 处理长方形养鱼池
8       RectPool    rObj;              // 定义引用
9       rObj = new RectPool();     // 创建长方形鱼池对象
10      rObj.a = sc.nextDouble();   // 输入长宽值
11      rObj.b = sc.nextDouble();
12      totalCost += rObj.RectCost( );   // 汇总造价
13      // 处理清水池和污水池
14      CirclePool    cObj1, cObj2;   // 定义引用
15      cObj1 = new CirclePool();   // 创建清水池对象
16      cObj2 = new CirclePool();   // 创建污水池对象
17      cObj1.r = sc.nextDouble();   // 输入清水池半径
18      cObj2.r = sc.nextDouble();   // 输入污水池半径
19      totalCost += cObj1.CircleCost( );   // 汇总造价
20      totalCost += cObj2.CircleCost( );   // 汇总造价
21      // 显示总造价 totalCost
22      System.out.println( totalCost );
23    }
24  }
```

程序员乙：长方形养鱼池类（RectPool.java）

```
public class RectPool {   // 长方形养鱼池类
    public double a, b;            // 字段：长宽
    public double RectCost( )   // 计算造价
    {   return (a*b *10);   }
}
```

程序员乙：圆形水池类（CirclePool.java）

```
public class CirclePool {   // 圆形水池类
    public doubler;                 // 字段：半径
    public double CircleCost( )   // 计算造价
    {   return (3.14 * r*r *10);   }
}
```

例 3-8 一个钟表类 Clock 的 Java 示意代码（Clock.java）

```
1  import java.util.Scanner;  // 导入外部程序 Scanner
2
3  public class Clock{            // 定义钟表类 Clock
4      // 将字段设为 private 权限，即私有成员
5      private   int hour;        // 字段 hour：保存小时数
6      private   int minute;      // 字段 minute：保存分钟数
7      private   int second;      // 字段 second：保存秒数
8      // 将方法设为 public 权限，即公有成员
9      public   void set( ) {    // 不带参数的方法 set()：从键盘输入时间
10         Scanner sc = new Scanner( System.in );      // 创建键盘扫描器对象 sc
11         hour = sc.nextInt();   minute = sc.nextInt();   second = sc.nextInt();
12     }
13     public   void set(int h, int m, int s) {          // 带参数的方法 set()：用参数设定时间
14         hour = h;   minute = m;   second = s;   // 将参数分别赋值给对应的数据成员
15     }
16     public   void show( ) {                                          // 方法 show：显示时间
17         System.out.println( hour +”:” +minute +”:” +second );  // 显示格式：时:分:秒
18     }
19 }
```

例 3-10 一个为钟表设置整点时间的 Java 示例代码（ClockTest.java）

```
1  import java.util.Scanner;  // 导入外部程序 Scanner
2
3  public class ClockTest {                          // 主类
4      public static void main(String[] args) {   // 主方法
5          int hour;   // 局部变量：普通变量，未初始化
6          Clock c1;   // 局部变量：引用变量，未初始化
7          hour = 12;      c1 = new Clock( );   // 为局部变量赋值
8          c1.set( 8, 30, 15 );   // 设置钟表对象 c1 的时间
9          c1.show( );          // 显示结果：8:30:15
10
11         Clock c2;                    // 再定义一个局部引用变量 c2
12         c2 = setHour( c1, hour );   // 调用 setHour()将 c1 设为整点，并将返回值赋值给 c2
13         c2.show( );   // 显示 c2 的时间，结果应为：12:0:0
14         c1.show( );   // 显示 c1 的时间，结果也为：12:0:0
15     }
16
17     private static Clock setHour( Clock rc, int h ) {   // 将钟表时间设为整点的方法
18         rc.set( h, 0, 0 );   // 设置 rc 的时间，小时数为接收的参数 h，分钟和秒数设为 0
19         return rc;
20     }
21 }
```

例 3-13 通过静态成员实现对钟表类 Clock 进行对象计数的 Java 示意代码（Clock.java）

```
1  public class Clock {   // 定义钟表类 Clock
2      public static   int totalClock = 0;   // 定义一个静态字段，记录已创建的 Clock 对象个数
3      privatestatic   void plusObj( ) {   totalClock ++;   }   // 定义一个静态方法，将计数加 1
4
5      private int hour,minute, second;          // 字段成员
6      public void set( ) { …… }                 // 不带参数的设置时间方法 set（代码省略）
7      public void set(int h, int m, int s) { …… }   // 带参数的设置时间方法 set（代码省略）
8      public void show( ) { …… }                // 显示时间方法 show（代码省略）
9      public Clock( )              // 定义一个构造方法
10     {   ……;   plusObj( );   }   // 通过构造方法为钟表对象增加计数功能
11 }
```

例 3-14 一个使用数学类 Math 中静态成员的演示程序

```
1  public class MathTest {   // 测试类：测试 Java 语言中数学类 Math 的静态成员
2
3      public static void main(String[] args) {                    // 主方法
4          System.out.println( "random()= " +Math.random() );   // 随机数函数（静态方法）
5          System.out.println( "random()= " +Math.random() );
6
7          System.out.println( "sqrt(36)= " +Math.sqrt(36) );      // 求平方根函数（静态方法）
8          System.out.println( "sin(30)= " +Math.sin(30 *Math.PI/180) );   // 正弦函数（静态方法）
9      }
10 }
```

例 3-15 一个遍历数组的 Java 演示程序（ArrayDemo.java）

```
1  public class ArrayDemo {                    // 主类
2      public static void main(String[] args) {   // 主方法
3          int iArray[ ] = { 2, 4, 6, 8, 10 };      // 定义一个 int 型数组 iArray，定义时初始化
4          for (int n = 0; n < iArray.length; n++)   // 遍历数组，逐个显示各数组元素的值
5              System.out.println( iArray[n] );   // 显示第 n 个元素的值
6          // 遍历数组，计算各数组元素的累加和
7          int sum = 0;
8          for (int n = 0; n < iArray.length; n++)
9              sum += iArray[n];   // 累加第 n 个数组元素
10         System.out.println( sum );
11
12         char cArray[ ] = { 'C', 'h', 'i', 'n', 'a' };      // 定义一个字符型数组 cArray，定义时初始化
13         for (int n = 0; n < cArray.length; n++) {   // 遍历数组，将所有小写字母改为大写
14             if (cArray[n] >= 'a' && cArray[n] <= 'z')   // 检查第 n 个元素是否小写字母
15                 cArray[n] -= 32;                    // 如果是，则将小写字母改为大写
16         }
17         System.out.println( cArray );   // 只有字符数组才能整体输出，显示结果：CHINA
```

```
18  }  }
```

例 3-16 一个遍历数组（增强 for 语句）的 Java 演示程序（EnhancedFor.java）

```
1   public class EnhancedFor {                    // 主类
2       public static void main(String[] args) {  // 主方法
3           int iArray[ ] = { 2, 4, 6, 8, 10 };       // 定义一个 int 型数组 iArray，定义时初始化
4           for (int x: iArray)   // 增强 for 语句：依次将数组 iArray 中的元素取出来，赋值给 x
5               System.out.println( x );   // 显示所取出的值
6           // 遍历数组，计算各数组元素的累加和
7           int sum = 0;
8           for (int x: iArray)   // 增强 for 语句：计算各数组元素的累加和
9               sum += x;       // 累加所取出的值
10          System.out.println( sum );
11
12          char cArray[ ] = { 'C', 'h', 'i', 'n', 'a' };   // 定义一个字符型数组 cArray，定义时初始化
13          for (char x: cArray) {   // 增强 for 语句：依次将数组 cArray 中的元素取出来，赋值给 x
14              if (x>= 'a' &&x<= 'z')   // 检查所取出的字符是否小写字母
15                  x -= 32;   // 将小写字母改为大写。注：此处只能修改 x，无法修改数组元素
16          }
17          System.out.println( cArray );   // 显示结果：China
18          // 可以看出，当需要修改数组元素时，还是只能用普通 for 语句
19  }  }
```

例 3-17 一个具有可变长形参的求最大值方法 Java 演示程序（VarArgument.java）

```
1   public class VarArgument {                    // 主类
2       public static int max(int...varArgs) {   // 具有可变长形参的求最大值方法
3           // 可变长形参 varArgs 所接收到的实参是以数组形式存放的，varArgs 是一个数组
4           if (varArgs.length< 1)   return 0;   // 如果没有传递实参，则直接返回 0
5           int result = varArgs[0];              // 先假设第 0 个元素就是最大值
6           for (int n = 1; n < varArgs.length; n++) {   // 求数组元素中的最大值
7               if (varArgs[n] > result)   result = varArgs[n];
8           }
9           /* 也可使用以下的增强 for 语句来求最大值
10          for (int e : varArgs)
11          {   if (e> result)   result = e;   }
12          */
```

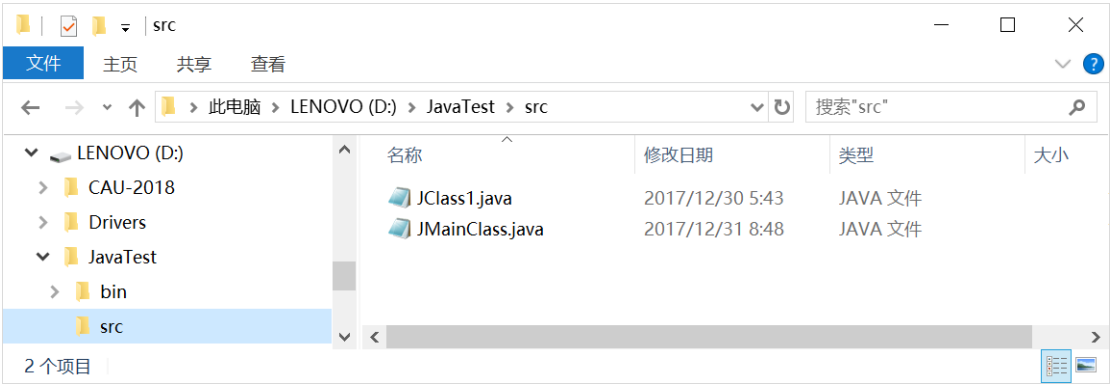
```
13              return result;
14          }
15
16          public static void main(String[] args) {        // 主方法
17              System.out.println( max(2, 4) );        // 传递 2 个实参，显示结果：4
18              System.out.println( max(2, 4, 6) );     // 传递 3 个实参，显示结果：6
19              System.out.println( max(2, 4, 6, 8) );  // 传递 4 个实参，显示结果：8
20              System.out.println( max(2) );           // 传递 1 个实参，显示结果：2
21              System.out.println( max() );            // 不传递实参，显示结果：0
22          }
23      }
```

例 3-18 一个二维数组的 Java 演示程序（Array2D.java）

```
1   public class Array2D {                          // 主类
2       public static void main(String[] args) {   // 主方法
3           // 定义一个 2 行 3 列的二维数组 a（可认为是一个矩阵），定义时初始化
4           int a[ ][ ] = { { 1, 3, 5 }, { 2, 4, 6 } };
5           // 数组遍历：按先行后列的次序显示各数组元素的值，需使用两重循环
6           for (int m = 0; m < a.length; m++) {        // 行循环：m 保存行下标
7               for (int n = 0; n <a[m].length; n++)    // 列循环：n 保存列下标
8                   System.out.print( a[m][n] + " " );
9               System.out.println( );   // 换行显示下一行数组元素
10          }
11          // 将二维数组（矩阵）a 转置后保存到 b 中
12          int b[ ][ ] = new int[3][2];   // 定义 3 行 2 列的二维数组 b
13          // 数组遍历：计算转置矩阵 b
14          for (int m = 0; m <b.length; m++) {         // 行循环：m 保存行下标
15              for (int n = 0; n <b[m].length; n++) {  // 列循环：n 保存列下标
16                  b[m][n] = a[n][m];   // 将 a 和 b 的行列下标互换后赋值，这就是矩阵的转置
17                  System.out.print( b[m][n] + " " );
18              }
19              System.out.println( );   // 换行显示下一行数组元素
20          }
21  }  }
```

例 3-19 一个对象数组的 Java 演示程序（ArrayObject.java）

```
1  public class ArrayObject {                    // 主类
2      public static void main(String[] args) {  // 主方法
3          Clock c[ ] = new Clock[6];            // 定义一个包含 6 个元素的钟表对象数组
4          // 遍历数组：创建钟表对象，设置并显示其时间。各钟表对象的时差为 1 小时
5          for (int n = 0; n < c.length; n++) {
6              c[n] = new Clock( );  // 创建钟表对象，将其引用赋值给第 n 个元素
7              c[n].set(n, 0, 0);    // 设置钟表对象的时间
8              c[n].show();          // 显示钟表对象的时间
9          }
10 } }
```

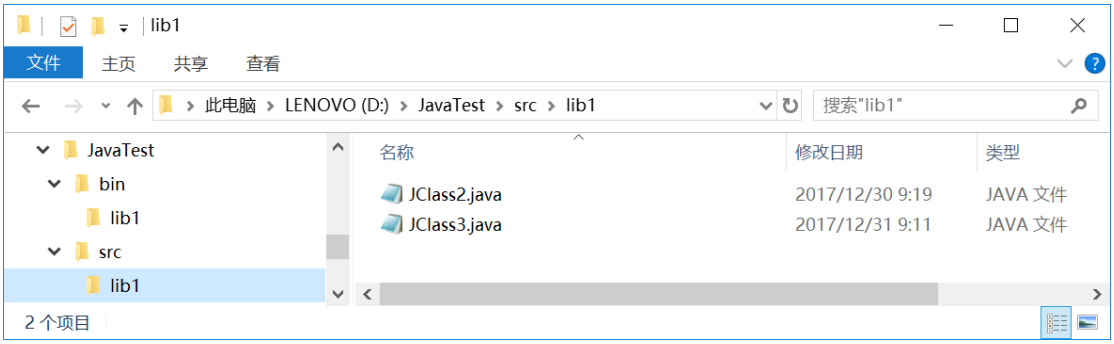


例 3-20 在项目 Project1 中添加两个类：JClass1 和 JMainClass

```
1  // 类 JClass1：源程序文件 JClass1.java
2
3  public class JClass1 {
4      private int f1 = 10;        // 一个字段
5      public void show1( ) {   // 一个方法
6          System.out.println( "JClass1: " +f1 );
7      }
8  }
```

```
// 主类 JMainClass：源程序文件 JMainClass.java
public class JMainClass {   // 主类

    public static void main(String[ ] args) {
        JClass1 obj = new JClass1( );
        obj.show1();
    }
}
```



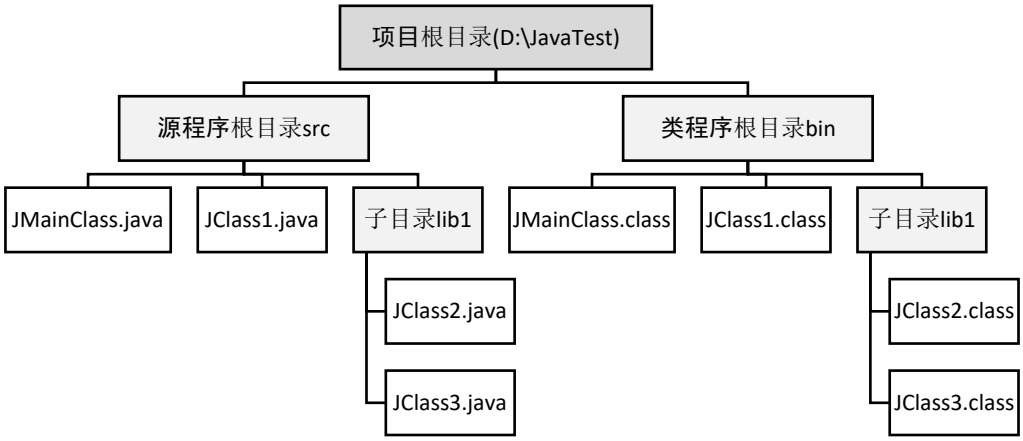
例 3-21 在项目 Project1 的包 lib1 中添加两个类：JClass2 和 JClass3

```
// 类 JClass2：源程序文件 lib1\JClass2.java
package lib1;  // 向编译器声明包名

public class JClass2 {
    private int f2 = 20;        // 一个字段
    public void show2( ) {   // 一个方法
        System.out.println( "JClass2: " +f2 );
    }
}
```

```
// 类 JClass3：源程序文件 lib1\JClass3.java
package lib1;  // 向编译器声明包名

public class JClass3 {
    private int f3 = 50;        // 一个字段
    public void show3( ) {   // 一个方法
        System.out.println( "JClass3: " +f3 );
    }
}
```



例 3-22 一个使用不同包里类的 Java 演示程序（JMainClass.java）

```
import lib1.JClass2;  // 预先导入包 lib1 中的类 JClass2
import lib1.JClass3;  // 预先导入包 lib1 中的类 JClass3
//import lib1.*;  // 或者预先导入包 lib1 中的所有类

public class JMainClass {                    // 主类
    public static void main(String[ ] args) {  // 主方法
        JClass1 obj1 = new JClass1( );       // 使用同一包里的类 JClass1，直接使用类名
        obj1.show1();
        // 下面使用 lib1 包里的类 JClass2、JClass3
        JClass2 obj2 = new JClass2( );  // 因为预先导入了 lib1 包里的类，这里可直接使用类名
        JClass3 obj3 = new JClass3( );
        /* 如果不预先导入 lib1 包，则需在类名之前加上包名。例如，
        lib1.JClass2 obj2 = new lib1.JClass2( );  // 使用 lib1 包里的类，需在类名前加上包名
        lib1.JClass3 obj3 = new lib1.JClass3( );
        */
        obj2.show2();      obj3.show3();
} }
```

第 4 章 面向对象程序设计之二

例 4-1 一个钟表类 Clock 的完整定义代码（Clock.java）

```
1  public class Clock {                          // 定义钟表类 Clock
2      private   int hour, minute, second;   // 字段：保存时分秒数据
3      public    void set(int h, int m, int s)    // 方法：设置钟表对象的时间
4      {   hour = h;   minute = m;   second = s;   }
5      public    void show( )                    // 方法：显示时间，显示格式：时:分:秒
6      {   System.out.println( hour +":" +minute +":" +second );   }
7
8      public    Clock( )                        // 无参构造方法：将时分秒数据都设为 0
9      {   hour = 0;   minute = 0;   second = 0;   }
10     public    Clock(int h, int m, int s)   // 有参构造方法：根据参数设置时间
11     {   hour = h;   minute = m;   second = s;   }
12     public    Clock( Clock oldObj )      // 拷贝构造方法：复制已有对象的时分秒数据
13     {   hour = oldObj.hour;   minute = oldObj.minute;   second = oldObj.second;   }
14 }
```

例 4-2 一个使用钟表类 Clock 组合出的双时区钟表类 DualClock（DualClock.java）

```
1  public class DualClock {   // 双时区钟表类：含有对象字段，属于组合类
2      public Clock c1, c2;   // 对象字段：两个 Clock 类的钟表对象，设为公有成员
3      public void setDual(int h, int m, int s)          // 设置方法：按参数设置 c1、c2 的时间
4      {   c1.set( h, m, s );   c2.set( h +1, m, s);   }   // 假设设为两个连续的时区
5      public void showDual( )   // 显示两个钟表的时间
6      {   c1.show( );   c2.show( );   }
7
8      public DualClock( ) {      // 组合类需要定义自己的构造方法
9          c1 = new Clock( );   // 组合类构造方法需使用运算符 new 创建对象字段
10         c2 = new Clock( );
11     }
12 }
```

例 4-3 对钟表类 Clock 重新包装得到一个带日历功能的包装类 DateClock

```
1  public class DateClock {   // 包装类（DateClock.java）
2      private Clock c;        // 对象字段：被包装的原始钟表（Clock）对象 c
3      // 以下代码都是为了对钟表对象 c 进行包装，为其增加日历功能
4      private int year, month, day;   // 添加字段：保存年月日数据
5      public void setDate(int y, int m, int d)   // 方法：设置日期
6      {   year = y;   month = m;   day = d;   }
7      public void show() {   // 方法：显示日期和时间
8          System.out.print(year +"-" +month +"-" +day +" ");   // 先显示日期
9          c.show();   // 再显示时间
10     }
11     public Clock getClock()          // 方法：获得包装前的原始钟表对象 c
12     {   return c;}
13     public DateClock(Clock obj)   // 构造方法：传递被包装的钟表对象
14     {   c = obj;   }   // 对象字段 c 直接引用传递过来的钟表对象 obj
15 }
```

```
1  public class DateClockTest {   // 主类（DateClockTest.java）
2
3      public static void main(String[] args) {        // 主方法
4          Clock cObj = new Clock(10, 30, 15 );   // 定义一个钟表对象 cObj
5          // 对钟表对象 cObj 进行包装，得到一个带日历的钟表对象 dcObj
6          DateClock dcObj = new DateClock( cObj );
7          dcObj.setDate(2018, 9, 1);   // 设置 dcObj 的日期
8          dcObj.show();   // 显示 dcObj 的日期和时间，显示结果：2018-9-1 10:30:15
9      }
10 }
```

例 4-4 通过继承与扩展钟表类 Clock 所定义出的手表类 Watch（Watch.java）

```
1  public class Watch extends Clock {   // 继承超类 Clock，在此基础上扩展出子类 Watch
2      private int band;                  // 新添加的字段：表带类型，1-金属，2-皮革
3      public void setBand(int b)      // 新添加的方法：设置表带类型
4      {   band = b;   }
5      public void show( ) {   // 重新定义显示时间的方法，显示格式：（表带类型）时:分:秒
6          // 先显示表带类型
7          if (band == 1)System.out.print("（金属表带）");
8          elseSystem.out.print("（皮革表带）");
9          // 再显示时间：调用超类的方法 show()
10         super.show();   // 关键字 super 表示超类
11     }
12 }
```

例 4-5 一个关于保护权限的 Java 演示程序

程序员甲：定义类 A（A.java）

```
1  public class A {
2      public   int x;       // 公有权限
3      private   int y;      // 私有权限
4      protected   int z;    // 保护权限
5      public   void aFun( ) {   // 在本类中访问，不受访问权限控制
6          x = 10;   // 访问公有成员，正确
7          y = 10;   // 访问私有成员，正确
8          z = 10;   // 访问保护成员，正确
9      }
10 }
```

程序员乙：使用类 A 定义对象（ATest.java）

```
1  public class ATest {   // 测试类
2      public static void main(String[] args) {
3          // 在其他类（非 A 的子类）中访问
4          A aObj = new A( );   // 先定义对象
5          aObj.x = 10;   // 访问公有成员，正确
6          aObj.y = 10;   // 访问私有成员，错误
7          // 如果与类 A 不在同一包中
8          aObj.z = 10;   // 访问保护成员，错误
9          // 如果与类 A 在同一包中
10         aObj.z = 10;   // 访问保护成员，正确
11     }
12 }  // 场合一：保护权限=默认权限
```

程序员丙：使用类 A 定义子类 B（B.java）

```
1  public class B extends A {
2      public   void bFun( ) {   // 在子类 B 中访问
3          x = 10;   // 访问公有超类成员，正确
4          y = 10;   // 访问私有超类成员，错误
5          z = 10;   // 访问保护超类成员，正确
6          // 子类可以访问保护权限的超类成员
7          // 不管子类与超类在不在同一包中
8      }
9  }
10
11
12 // 场合二：保护权限=为子类定向开放的权限
```

例 4-6 为子类 Watch 定义的构造方法示例代码

```
1  public class Clock {   // 钟表类 Clock（超类）
2      ……   // 这里只节选例 4-1 中的构造方法，其他代码省略
3      public   Clock( )   // 无参构造方法：将时分秒字段都设为 0
4      {   hour = 0;   minute = 0;   second = 0;   }
5      public   Clock(int h, int m, int s)   // 有参构造方法：根据参数设置时间
6      {   hour = h;   minute = m;   second = s;   }
7      public   Clock( Clock oldObj )    // 拷贝构造方法：复制已有对象的时分秒字段
8      {   hour= oldObj.hour;minute = oldObj.minute;second = oldObj.second;   }
9  }
```

```
1  public class Watch extends Clock {   // 手表类 Watch（子类）
2      ……   // 这里列出为子类 Watch 定义的构造方法，其他代码省略
3      public   Watch( ) {   // 无参构造方法
4          super( );         // 先调用超类 Clock 的无参构造方法，初始化超类字段（时分秒）
5                            // 也可以调用超类 Clock 的有参构造方法：super( 0, 0, 0 );
6          band = 1;         // 然后再初始化子类字段：表带类型，直接对其赋值
7      }
```

```
8       public  Watch( int h, int m, int s, int b ) {  // 有参构造方法：初始化时分秒和表带类型
9           super( h, m, s );  // 需调用超类 Clock 的有参构造方法，初始化超类字段（时分秒）
10          band = b;          // 然后再初始化子类字段：表带类型，直接对其赋值
11      }
12      public  Watch( Watch oldObj ) {  // 拷贝构造方法
13          super( oldObj );         // 先调用超类 Clock 的拷贝构造方法，初始化超类字段
14          band = oldObj.band;  // 然后再初始化子类字段：表带类型，直接对其赋值
15      }
16  }
```

例 4-7 一个初始化子类中字段成员的 Java 演示程序

超类 Sup（Sup.java）

```
1   class Sup {  // 定义超类 Sup
2     public      int x;  // 公有成员
3     private     int y;  // 私有成员
4     protected   int z;  // 保护成员
5     public  Sup( ) {  // 构造方法
6       // 在构造方法中显示创建过程
7       System.out.println("Sup enter: " +x +y +z);
8       x = 1;   y = 1;   z = 1;
9       System.out.println("Sup exit: " +x +y +z);
10    }
11  }
12
13
```

子类 Sub（Sub.java）

```
class Sub extends Sup {  // 定义子类 Sub
  private int a = 2;  // 新添加的成员，初始化②
  public  Sub( ) {  // 子类的构造方法
    super( );         // 初始化①
    // 在构造方法中显示创建过程
    System.out.println("Sub enter: " +x +"?" +z +a);
    a = 3;              // 初始化③
    x = 3;
    // y = 3;  // 子类不能访问私有的超类成员
    z = 3;
    System.out.println("Sub exit: " +x +"?" +z +a);
  }
}
```

```
1   public class FieldInitDemo {                  // 测试类（FieldInitDemo.java）
2       public static void main(String[] args) {  // 主方法
3           Sub obj = new Sub( );                 // 创建子类 Sub 的对象，将调用子类的构造方法
4       }
5   }
```

例 4-8 一个处理钟表类 Clock 对象的方法 setGMT()及其测试类 GMTTest（GMTTest.java）

```
1  public class GMTTest {                          // 测试类
2      public static void main(String[] args) {    // 主方法
3          Clock cObj = new Clock( );               // 创建一个钟表对象 cObj
4          // 给定 GMT 时间 8:30:15，调用方法 setGMT()将其转成北京时间后再设置给 cObj
5          setGMT(cObj, 8, 30, 15);
6      }
7
8      // 处理钟表类 Clock 对象的方法 setGMT()：
9      // 给定 GMT 时间，先将其转换成北京时间，然后再设置给钟表对象 obj
10     public static    void setGMT(Clock obj, int hGMT, int mGMT, int sGMT) {
11         int h, m, s;       // 先定义 3 个保存北京时间的变量
12         h = hGMT +8;   // 北京时间比 GMT 时间早 8 个小时，即小时数加 8
13         m = mGMT;    s = sGMT;
14         obj.set(h, m, s);   // 将转换后的北京时间设置给钟表对象 obj
15         obj.show();        // 显示时间：GMT 时间 8:30:15 所对应的北京时间是 16:30:15
16 }  }
```

例 4-9 从钟表类 Clock 扩展出的的 3 个子类

手表类 **Watch**

```
1  class Watch extends Clock {   // 手表类
2     public int band = 1;     // 新添加表带类型
3     public void show( ) {   // 重写 show 方法
4        if (band == 1)   // 金属表带
5           System.out.print( “(金属表带)” );
6        else              // 皮革表带
7           System.out.print( “(皮革表带)” );
8        super.show();
9  }  }
```

潜水表类 **DivingWatch**

```
1  class DivingWatch extends Watch{   // 潜水表类
2     public int depth = 10;     // 新添加最大深度
3        public void show( ) {   // 重写 show 方法
4           System.out.print( “(“+depth+”米)” );
5        super.show();
6  }  }
```

挂钟类 **WallClock**

```
class WallClock extends Clock {   // 挂钟类
   public int size = 12;     // 新添加表盘尺寸
   public void show( ) {   // 重写 show 方法
      System.out.print( “(“ +size +”英寸)” );
      super.show();
}  }
```

例 4-11 抽象超类（学生类）后再扩展本科生类和研究生类的 Java 示意代码

Student：**学生类**（抽象出的超类）

```
1  public class Student {              // 超类：学生类
2     public char Name[], ID[];        // 姓名、学号
3     public int Age;                  // 年龄
4     public float Score;              // 课堂成绩
5     public void Input() { … }        // 输入学生基本信息
6     public void ShowInfo() { … }     // 显示学生基本信息
7  }
```

Undergraduate：**本科生类**（子类）

```
1  public class Undergraduate extends Student {
2     public float PracticeScore;    // 毕业设计成绩
3
4     public float TotalScore() { … }  // 计算总成绩
5     public void Input() { … }        // 重写输入方法
6     public void ShowInfo() { … }     // 重写显示方法
7  }
```

Graduate：**研究生类**（子类）

```
1  class Graduate extends Student {
2     public double PaperScore;    // 毕业论文成绩
3     public int Thesis;           // 发表论文数量
4     public float TotalScore() { … }  // 计算总成绩
5     public void Input() { … }        // 重写输入方法
6     public void ShowInfo() { … }     // 重写显示方法
7  }
```

例 4-14 一个完整的儿童手表类 ChildrenWatch 定义（ChildrenWatch.java）及测试代码

```
1  public class ChildrenWatch extends Watch implements Callable, Positionable {
2        // 继承手表类 Watch，同时实现接口 Callable 和 Positionable
3        public void call(int number)                  // 实现接口 Callable 所规定的打电话方法
4        {   System.out.println("Call " +number);   }  // 显示一个模拟打电话的提示信息
5        public void answer( )                         // 实现接口 Callable 所规定的接电话方法
6        {   System.out.println("Answer a call");   }  // 显示一个模拟接电话的提示信息
7
8        public void showPosition( )  // 实现接口 Positionable 所规定的显示定位方法
9        {   System.out.println("Show position");   }  // 显示一个模拟定位的提示信息
10 }
```

```
1  public class CWatchTest {                     // 测试类（CWatchTest.java）
2        public static void main(String[] args) {   // 主方法
3              ChildrenWatch cw = new ChildrenWatch( );  // 使用儿童手表类定义对象
4              cw.set(8, 30, 15);       // 设置手表时间
5              cw.show( );              // 显示手表信息：（金属表带）8:30:15
6              cw.call(6789);   // 打电话。模拟显示结果：Call 6789
7              cw.answer( );    // 接电话。模拟显示结果：Answer a call
8              cw.showPosition( );   // 显示定位。模拟显示结果：Show position
9        }
10 }
```

例 4-15 一个包含内部类 B 的外部类 AwithInner 示例代码（AwithInner.java）

```
1  public class AwithInner {   // 外部类
2      public    int x = 10;     // 公有成员 x
3      private    int y = 20;   // 私有成员 y
4
5      public class B {   // 内部类
6          public int z = 30;
7          public void bShow() {
8              System.out.println(x);   // 内部类可以直接访问外部类的成员 x
9              System.out.println(y);   // 内部类可以访问外部类的私有成员 y
10             System.out.println(z);
11     }   }
12
13     public void aShow() {     // 在外部类中使用内部类 B
14         B bObj = new B();   // 与使用普通的类一样
15         bObj.bShow();
16 }   }
```

例 4-16 一个包含局部类 B 的外部类 AwithLocal 示例代码（AwithLocal.java）

```
1  public class AwithLocal {   // 外部类，其方法 aMethod 中包含一个局部类 B
2      private int a = 10;       // 外部类的私有成员 a
3      public void aMethod( int x) {   // 方法中的形参 x
4          final int y = 30;              // 方法中的局部只读变量（或称常量）y
5
6          class B {              // 定义在方法 aMethod 中的局部类
7              int b = 40;   // 局部类的成员 b
8              void showA( )   // 访问外部类的成员 a
9              {   System.out.println( a );   }
10             void showXY( )   // 访问方法中的形参 x 和局部只读变量 y
11             {   System.out.println( x +" and " +y );   }
12             void showB( )     // 访问本类的成员 b
13             {   System.out.println( b );   }
14         }
15
16         B obj = new B( );   // 创建局部类 B 的对象 obj
17         obj.showA( );   obj.showXY( );   obj.showB( );   // 调用对象 obj 的方法
18 }   }
```

例 4-17 一个实现接口的局部类 B 示例代码（在 4-16 的 AwithLocal.java 基础上修改而来）

```
1  public class AwithLocal {   // 外部类，其方法 aMethod 中包含一个局部类 B
2        private int a = 10;       // 外部类的私有成员 a
3        public void aMethod( int x) {   // 方法中的形参 x
4              final int y = 30;              // 方法中的局部只读变量（或称常量）y
5
6              class B implements IShow {   // 实现接口 IShow 的局部类 B
7                    int b = 40;                  // 局部类的成员 b
8                    public void show( )        // 实现接口 IShow 里的抽象方法 show
9                    {   System.out.println(a+" and " +b );   }
10             }
11
12             B obj = new B( );   // 创建局部类 B 的对象 obj
13             obj.show( );          // 调用对象 obj 的方法 show
14 }  }
15 public interface IShow {     // 接口 IShow
16       public void show( );   // 定义了一个抽象方法 show
17 }
```

例 4-18 一个实现接口的匿名类示例代码（在 4-17 的 AwithLocal.java 基础上修改而来）

```
1  public class AwithLocal {   // 外部类，其方法 aMethod 中包含一个局部类 B
2        private int a = 10;       // 外部类的私有成员 a
3        public void aMethod( int x) {   // 方法中的形参 x
4              final int y = 30;              // 方法中的局部只读变量（或称常量）y
5
6              IShow obj = new IShow( ) {   // 用一条语句完成定义匿名类和创建对象的工作
7                    int b = 40;                  // 匿名类的成员 b
8                    public void show( )       // 实现接口 IShow 里的抽象方法 show
9                    {   System.out.println(a+" and " +b );   }
10             };
11             obj.show( );   // 调用对象 obj 的方法 show
12 }  }
13 public interface IShow {     // 接口 IShow
14       public void show( );   // 定义了一个抽象方法 show
15 }
```

例 4-19 一个功能接口 IOperator 和功能类 AClass 的示例代码

	IOperator：功能接口	AClass：实现接口 IOperator 的功能类
1	public interface **IOperator** { // 功能接口	class **AClass** implements **IOperator** { // 功能类
2	int **operation**(int x, int y); // 抽象方法	int **operation**(int x, int y) // 实现接口里的方法
3	}	{ return x +y; } // 加法运算
4		}

第 5 章 Java 基础类库

例 5-1 一个数学类 Math 的测试程序示例代码（MathTest.java）

```
1  public class MathTest {                          // 测试类
2      public static void main(String[] args) {    // 主方法
3          System.out.println( Math.abs( -8 ) );          // 求绝对值
4          System.out.println( Math.sqrt( 16 ) );         // 求平方根
5          System.out.println( Math.sin( Math.PI/2 ) );   // 求正弦值
6          System.out.println( Math.toDegrees( Math.PI ) );  // 将弧度换算成角度
7          System.out.println( Math.random( ) );   // 取一个随机数
8          System.out.println( Math.random( ) );   // 再取一个随机数
9      }
10 }
```

例 5-2 一个字符串类 String 的测试程序示例代码（StringTest.java）

```
1  public class StringTest {                        // 测试类
2      public static void main(String[] args) {    // 主方法
3          int x = 5;    double y = 16.8;
4          String s = String.format("x= %d, y= %5.2f", x, y);   // 格式化字符串
5          System.out.println( s );   // 显示字符串 s
6          // 下面演示字符串对象的定义与处理
7          String s1 = new String( );   // 先定义 3 个字符串对象
8          String s2 = new String("Abcd");
9          String s3 = new String("Abcd cde");
10         System.out.println( s1.length() );   // 空字符串的长度为 0
11         System.out.println( s2.length() );   // s2 的长度为 4
12         System.out.println( s2.toUpperCase() );   // 返回一个大写的字符串
13         System.out.println( s3.indexOf("cd") );   // 返回 cd 第一次出现的位置
14         System.out.println( s3.substring(1, 3) );   // 返回 1~3 之间的子字符串
15         System.out.println( s3.concat("fg") );      // 连接：s3 +"fg"
16         System.out.println( s3 +"fg" );             // 连接：s3 +"fg “
17 }  }
```

例 5-3 一个可变字符串类 StringBuilder 的测试程序示例代码（SBuilderTest.java）

```
1  public class SBuilderTest {                          // 测试类
2      public static void main(String[] args) {   // 主方法
3          StringBuilder s = new StringBuilder( 100 );   // 字符容量为 100 个字符（单线程）
4          // StringBuffer s = new StringBuffer( 100 );   // 类 StringBuffer 可支持多线程
5          s.append("helloChina");                 // 追加字符
6          System.out.println( s.toString() );   // 显示所返回字符串内容
7          System.out.println( s.length() );     // 显示字符串长度
8          System.out.println( s.capacity() );   // 显示字符容量
9          s.setCharAt(0, 'H');           // 设定指定位置的字符
10         System.out.println( s );      // 可以省略“.toString()”
11         s.replace(5, 10, "中国");   // 将 5~10 之间的字符替换成“中国”
12         System.out.println( s );
13         System.out.println( s.length() );              // 显示新字符串长度
14         s.insert(5, ',');   System.out.println( s );   // 插入一个字符“,”，然后显示内容
15         s.delete(5, 8);   System.out.println( s );   // 删除 5~8 之间的字符，然后显示内容
16 }  }
```

例 5-4 一个整数类 Integer 的测试程序示例代码（WrapperTest.java）

```
1  public class WrapperTest {                          // 测试类
2      public static void main(String[] args) {   // 主方法
3          System.out.println(Integer.BYTES);            // int 型的字节数
4          System.out.println(Integer.MIN_VALUE);   // int 型的最小值
5          System.out.println(Integer.MAX_VALUE);   // int 型的最大值
6          // 下面演示比较两个 Integer 对象的大小
7          Integer iObj1 = new Integer(20);       // 初始化为 20
8          Integer iObj2 = new Integer("30");   // 初始化为 30
9          System.out.println( iObj1 > iObj2 );                  // 比较大小
10         System.out.println( iObj1.compareTo(iObj2) );   // 比较大小
11         // 下面演示 Integer 与其他类型之间的转换
12         int i = iObj2.intValue( );   System.out.println( i );   // 将 Interger 转成 int
13         Integer iObj3 = Integer.valueOf( i+10 );   // 将 int 转成 Integer
14         System.out.println( iObj3.toString() );     // 将 Integer 转成 String
15         i = Integer.parseInt("50");   System.out.println( i );   // 将字符串转成 int
16         System.out.println( Integer.toString(i) );                  // 将 int 转成字符串
17 }  }
```

例 5-6 一个重写 Object 方法的新钟表类 Clock 及其测试类的示例代码

```
class Clock implements Cloneable {   // 自动继承 Object 类，实现接口 Cloneable（Clock.java）
    // 此处省略例 4-1 中类 Clock 已有的代码，下面演示重写从 Object 类继承来的方法
    public String toString( )   // 重写方法 toString()
    {   return String.format("Clock@%d:%d:%d", hour, minute, second);   }
    public int hashCode( )   // 重写方法 hashCode()
    {   return second;   }   // 生成哈希码：简单地将描述作为钟表对象的哈希码
    public boolean equals(Object obj) {   // 重写方法 equals()
        if ((obj instanceof Clock) == false)   return false;   // 类型不同，则直接返回 false
        Clock c = (Clock)obj;   // 将 Object 类型转成 Clock 类型
        return c.hour == hour && c.minute == minute && c.second == second;
        // 比较两个钟表对象的时间，如果时、分、秒都相同则返回 true，否则返回 false
    }
    public Object clone( ) throws CloneNotSupportedException   // 重写方法 clone ()
    {   Clock c = (Clock)super.clone();   return c;   }   // 克隆一个钟表对象
    // 注：clone ()方法头后面的“throws ……”是 Java 语言的异常处理，将在后面讲解
}
```

```
public class ObjectTest {                          // 测试类（ObjectTest.java）
    public static void main(String[] args) {   // 主方法
        Clock cObj = new Clock(8, 30, 15);
        System.out.println( cObj );                 // 显示引用变量
        System.out.println( cObj.toString() );   // 转成字符串，使用重写的 toString
        System.out.println( cObj.getClass().getName() );   // 取得对象 cObj 的类名
        // 下面演示如何比较两个钟表对象是否相等
        Clock cObj1 = new Clock(8, 30, 15);   // 新建对象，设置与 cObj 相同的时间
        Clock cObj2 = cObj;                          // cObj2 与 cObj 引用同一钟表对象
        System.out.println( cObj1 == cObj );   // 检查引用是否相同，即是否引用了同一对象
        System.out.println( cObj2 == cObj );   // 检查两个引用是否相同
        System.out.println( cObj1.equals(cObj) );   // 检查两个对象的内容（时间）是否相同
        System.out.println( cObj.hashCode() );      // 显示 cObj 对象的哈希码
        System.out.println( cObj1.hashCode() );     // 显示 cObj1 对象的哈希码
        // 下面演示如何克隆一个钟表对象
        try {   // try-catch 是 Java 语言的异常处理，将在后面讲解
            Clock cObj3 = (Clock)cObj.clone();        // 克隆一个和 cObj 一样的对象
            System.out.println( cObj3.toString() );   // 检查克隆对象的内容是否相同
        }   catch(CloneNotSupportedException e) { };
}   }
```

例 5-6 一个重写 Object 方法的新钟表类 Clock 及其测试类的示例代码

```
class Clock implements Cloneable {   // 自动继承 Object 类，实现接口 Cloneable（Clock.java）
    // 此处省略例 4-1 中类 Clock 已有的代码，下面演示重写从 Object 类继承来的方法
    public String toString( )   // 重写方法 toString()
    {   return String.format("Clock@%d:%d:%d", hour, minute, second);   }
    public int hashCode( )   // 重写方法 hashCode()
    {   return hour*10000 + minute*100 + second;   }   // 生成哈希码
    public boolean equals(Object obj) {   // 重写方法 equals()
        if ((obj instanceof Clock) == false)   return false;   // 类型不同，则直接返回 false
        Clock c = (Clock)obj;   // 将 Object 类型转成 Clock 类型
        return c.hour == hour && c.minute == minute && c.second == second;   // 比较时分秒
        //return c.hashCode() == hashCode();                    // 本例中也可以比较哈希码
    }
    public Object clone( ) throws CloneNotSupportedException   // 重写方法 clone ()
    {   Clock c = (Clock)super.clone();   return c;   }   // 克隆一个钟表对象
    // 注：clone ()方法头后面的“throws …”是 Java 语言的异常处理，将在后面讲解
}
```

```
public class ObjectTest {                    // 测试类（ObjectTest.java）
    public static void main(String[] args) {   // 主方法
        Clock cObj = new Clock(8, 30, 15);
        System.out.println( cObj );              // 显示引用变量
        System.out.println( cObj.toString() );   // 转成字符串，使用重写的 toString
        System.out.println( cObj.getClass().getName() );   // 取得对象 cObj 的类名
        // 下面演示如何比较两个钟表对象是否相等
        Clock cObj1 = new Clock(8, 30, 15);   // 新建对象，设置与 cObj 相同的时间
        Clock cObj2 = cObj;                   // cObj2 与 cObj 引用同一钟表对象
        System.out.println( cObj1 == cObj );   // 检查引用是否相同，即是否引用了同一对象
        System.out.println( cObj2 == cObj );   // 检查两个引用是否相同
        System.out.println( cObj1.equals(cObj) );   // 检查两个对象的内容是否相同
        System.out.println( cObj.hashCode() );      // 显示 cObj 对象的哈希码
        System.out.println( cObj1.hashCode() );     // 显示 cObj1 对象的哈希码
        // 下面演示如何克隆一个钟表对象
        try {   // try-catch 是 Java 语言的异常处理，将在后面讲解
            Clock cObj3 = (Clock)cObj.clone();        // 克隆一个和 cObj 一样的对象
            System.out.println( cObj3.toString() );   // 检查克隆对象的内容是否相同
        }   catch(CloneNotSupportedException e) { };
}   }
```

例 5-7 一个简单的 Java 除法运算程序（存在**语法错误**）（SyntaxError.java）

```
1  import java.util.Scanner;
2  public class SyntaxError {   // 一个存在语法错误的类
3      int Div(int n) {          // 方法功能：求 100÷n
4          int result;
5          result = 100 / n;   // 求 100÷n
6          return result;
7      }
8
9      public static void main(String[] args) {   // 主方法是一个静态方法
10         int N;
11         Scanner sc = new Scanner( System.in );   // 创建键盘扫描器对象
12         N = sc.nextInt( );            // 键盘输入 N 的值
13         int retValue = Div( N );   // 语法错误：调用非静态方法 Div 计算 100÷N
14         System.out.println( "100÷" +N +"= " +retValue );
15 }  }
```

例 5-8 另一个简单的 Java 除法运算程序（存在**语义错误**）（SemanticsError.java）

```
1  import java.util.Scanner;
2  public class SemanticsError {   // 一个存在语义错误的类
3      static int Div(int n) {       // 方法功能：求 100÷n
4          int result;
5          result = 100 * n;     // 语义错误：将除法错误写成了乘法，语法正确但语义错误
6          return result;
7      }
8
9      public static void main(String[] args) {   // 主方法
10         int N;
11         Scanner sc = new Scanner( System.in );   // 创建键盘扫描器对象
12         N = sc.nextInt( );            // 键盘输入 N 的值
13         int retValue = Div( N );   // 调用方法 Div 计算：100÷N
14         System.out.println( "100÷" +N +"= " +retValue );
15 }  }
```

例 5-10 一个添加了异常处理机制的 Java 除法运算程序（ErrorTryCatch.java）

```
1  import java.util.Scanner;
2  public class ErrorTryCatch {   // 一个添加了异常处理机制的类
3      static int Div(int n) {      // 方法功能：求 100÷n
4          int result;
5          if (n <= 0)        // 检查异常：如果 n<=0，则属于异常情况
6              throw ( new RuntimeException("输入的数值必须为正整数") );   // 报告异常
7          result = 100 / n;
8          return result;
```

```
9       }
10
11      public static void main(String[] args) {   // 主方法
12          int N;
13          Scanner sc = new Scanner( System.in );   // 创建键盘扫描器对象
14          N = sc.nextInt( );   // 键盘输入 N 的值
15          try {                   // 启用 Java 异常处理机制
16              int retValue = Div( N );   // 调用方法 Div 计算：100÷N
17              System.out.println( "100÷" +N +"= " +retValue );
18          }
19          catch(RuntimeException e)   // 捕获并处理异常
20          {   System.out.println( e.getMessage() );   }
21  }   }
```

例 5-11 一个 Java 异常处理机制的演示程序（ExceptionTest.java）

```
1   import java.io.IOException;
2   public class ExceptionTest {        // 测试类
3       static void fun(int choice) {   // 根据参数 choice 模拟不同的异常，然后进行异常处理
4           System.out.println( "choice: " +choice );   // 显示提示信息，用于观察执行流程
5           System.out.println( "Before try-catch" );
6
7           try {
8               System.out.println( "Before throw" );
9               if (choice == 1)   // 1：模拟算术运算异常
10                  throw new ArithmeticException("ArithmeticException");
11              else if (choice == 2)   // 2：模拟输入输出异常
12                  throw new IOException("IOException");
13              System.out.println( "After throw" );
14          }
15          catch(ArithmeticException e)   // 捕获并处理算术运算异常
16          {   System.out.println( e.toString() );   }
17          catch(IOException e)   // 捕获并处理输入输出异常
18          {   System.out.println( e.toString() );   }
19          finally   // 善后处理
20          {   System.out.println( "finally block" );   }
21
22          System.out.println( "After try-catch" );
23      }
24
25      public static void main(String[] args)   // 主方法
26      {   fun( 1 );   }   // 通过不同实参来模拟不同的异常，例如 fun(1)、fun(2)、fun(0)
27  }
```

例 5-14 一个 T 类型的泛型集合类示例代码

```
1  class GenericSet<T> {   // 泛型集合类：类型形参 T 可以指代任意一种具体的数据类型
2          public T set[];                 // 用于存放 T 类型数据的数组
3          public GenericSet(T p[])    // 构造方法
4          {    set = p;    }
5          public void show() {         // 显示数据集合中的元素
6                  for (int n = 0; n < set.length; n++)
7                          System.out.print( set[n] +" " );
8                  System.out.println();
9  }   }
```

例 5-17 一个存储和处理学生成绩单的 Java 示例代码（StudentScoreTest.java）

```
1  public class StudentScoreTest {                    // 主类
2          public static void main(String[] args) {     // 主方法
3                  Student sa[] = new Student[3];       // 创建一个保存 3 名学生成绩的对象数组
4                  sa[0] = new Student( "张三", 92 );   // 添加数据元素
5                  sa[1] = new Student( "李四", 86 );
6                  sa[2] = new Student( "王五", 95 );
7                  for (int n = 0; n < sa.length; n++)    // 遍历数组，显示学生成绩单
8                          System.out.println( sa[n].toString() );
9  }   }
10
11 class Student {    // 学生成绩类
12         private String name;    // 姓名
13         private int score;          // 成绩
14         public Student(String p1, int p2)    // 构造方法
15         {    name = p1;    score = p2;    }
16         public String toString()                // 重写 toString()方法
17         {    return String.format("%s: %d", name, score);    }
18         public boolean equals(Object obj) {    // 重写 equals()方法
19                 if ((obj instanceofStudent) == false)    return false;    // 类型不同，则直接返回 false
20                 Student s = (Student)obj;
21                 return name.equals(s.name);    // 比较姓名是否相同
22 }   }
```

例 5-18 一个使用类 ArrayList<E>创建数组列表的 Java 示例代码（ArrayListTest.java）

```
1  import java.util.ArrayList;      // 导入数组列表类 ArrayList
2  public class ArrayListTest {    // 测试类
3         public static void main(String[] args) {    // 主方法
4                 ArrayList<Integer> v1 = new ArrayList<Integer>( );         // Integer 型数组列表
5                 v1.add( 3 );    v1.add( 9 );    v1.add( 7 );    v1.add( 5 );    // 添加元素
6                 for (int n = 0; n < v1.size(); n++)    // 使用序号（下标）遍历显示各元素
7                         System.out.print( v1.get(n) +", " );
```

```
        System.out.println();

        ArrayList<Student> v2 = new ArrayList<>( );   // Student 型数组列表
        v2.add( new Student("张三", 92) );            // 添加元素
        v2.add( new Student("李四", 86) );
        v2.add( new Student("王五", 95) );
        for (int n = 0; n < v2.size(); n++)   // 使用序号（下标）遍历显示各元素
            System.out.println( v2.get(n) );
} }
```

例 5-19 一个使用 Collections 类中静态方法处理数组列表的 Java 示例代码（ArrayListTest.java）

```
import java.util.ArrayList;      // 导入数组列表类 ArrayList
import java.util.Collections;   // 导入集合算法类
public class ArrayListTest {                    // 测试类
    public static void main(String[] args) {   // 主方法
        ArrayList<Integer> v1 = new ArrayList<>( );              // Integer 型数组列表
        v1.add( 3 );   v1.add( 9 );   v1.add( 7 );   v1.add( 5 );   // 添加元素
        System.out.println( v1 );   // 直接显示整个数组列表
        Collections.sort( v1 );      // 对元素进行排序
        System.out.println( v1 );
        Collections.reverse( v1 );   // 逆序排列所有元素
        System.out.println( v1 );
        System.out.println( Collections.max( v1 ) );   // 求数据集合中元素的最大值
} }
```

例 5-20 一个使用类 linkedList<E>实现队列和堆栈功能的 Java 示例代码（LinkedListTest.java）

```
import java.util.LinkedList;   // 导入双端队列类 LinkedList
public class LinkedListTest {                   // 测试类
    public static void main(String[] args) {   // 主方法
        int n;
        // 下面演示使用 Integer 型双端队列实现一个队列
        LinkedList<Integer> q = new LinkedList<>();   // Integer 型双端队列
        for (n = 1; n <= 3; n++)
            q.offer( n );   // 实现一个队列：在队尾添加一个元素
        System.out.println( q );   // 显示队列
        for (n = 1; n <= 3; n++)
            System.out.print( q.poll() +"," );   // 取出并删除队列头的元素
        System.out.println( "\n" +q +"\n" );    // 再次显示队列，此时队列为空
        // 下面演示使用 Integer 型双端队列实现一个堆栈
        LinkedList<Integer> s = new LinkedList<>();   // Integer 型双端队列
        for (n = 1; n <= 3; n++)
            s.push( n );            // 实现一个堆栈：向栈中压入一个元素
        System.out.println( s );   // 显示堆栈
```

```
18              for (n = 1; n <= 3; n++)
19                  System.out.print( s.pop() +"," );   // 从栈中弹出一个元素
20              System.out.println( "\n" +s );          // 再次显示堆栈，此时堆栈为空
21  }   }
```

例 5-21 一个使用类 HashSet<E>实现集合功能的 Java 示例代码（HashSetTest.java）

```
1   import java.util.HashSet;   // 导入集合类 HashSet
2   public class HashSetTest {                      // 测试类
3       public static void main(String[] args) {   // 主方法
4           HashSet<String> s = new HashSet<>( );       // String 型集合
5           s.add("1st");   s.add("2nd");   s.add("3rd");   // 添加元素
6           s.add("4th");   s.add("5th");
7           System.out.println( s );   // 显示集合，各元素按其哈希码的顺序存放
8           s.remove("4th");           // 删除元素
9           System.out.println( s );   // 再次显示集合
10          System.out.println( s.size() );   // 显示集合中的元素个数
11  }   }
```

例 5-22 使用映射类 HashMap<K, V>存储表 5-4 中学生成绩单的 Java 示例代码（HashMapTest.java）

```
1   import java.util.HashMap;   // 导入映射类 HashMap
2   import java.util.Set;       // 导入集合类 HashSet
3   public class HashMapTest {                      // 测试类
4       public static void main(String[] args) {   // 主方法
5           HashMap<String, Integer> h = new HashMap<>();     // String-Integer 型映射
6           h.put( "张三", 92 );   h.put( "李四", 86 );   h.put( "王五", 95 );   // 添加元素
7           // 遍历映射：取出键的集合，然后遍历该集合
8           Set<String> kSet = h.keySet( );   // 取出映射对象 h 中键的集合
9           for (String k: kSet)                    // 使用增强 for 语句遍历键的集合 kSet
10          {   System.out.println( k +" - " +h.get(k) );   }   // 取出映射 h 中键 k 所对应的值
11  }   }
```

例 5-23 一个完整的枚举类型 WeekDay 定义及使用示例代码（EnumTest.java）

```
1   public class EnumTest {                         // 测试类
2       public static void main(String[] args) {   // 主方法
3           for ( WeekDay e: WeekDay.values() )     // 列出 WeekDay 中的所有枚举常量
4               System.out.print( e.name() +", " );   // 显示枚举常量的名称
5           System.out.println();
6           WeekDay d = WeekDay.MONDAY;   // 定义枚举变量并初始化为 MONDAY
7           System.out.println( d.ordinal() );    // 显示 MONDAY 的内部整数编号
8           System.out.println( d.name() );       // 显示 MONDAY 的名称
9           d.isWeekend( );                       // 检查 MONDAY 是否周末
10  }   }
11
```

```
enum WeekDay {   // 定义一个表示星期几的枚举类型 WeekDay
      SUNDAY, MONDAY, TUESDAY, WEDNESDAY,THURSDAY, FRIDAY, SATURDAY;   // 枚举常量
      public   void isWeekend() {                          // 添加方法成员：检查是否周末
            if (this == SATURDAY || this == SUNDAY)   // 枚举类型可以做关系运算
                  System.out.println( "The day is Weekend." );
            else
                  System.out.println( "The day is not Weekend." );
}  }
```

例 5-24 一个添加了文档注释的钟表类 DocClock 示例代码（DocClock.java）

```
/**
 * 类的功能简介：DocClock 是一个钟表类，其中添加了文档注释。
 */
public class DocClock {   // 添加了文档注释的钟表类
      private int hour, minute, second;   // 私有成员不对外开放，通常不需要添加文档注释
      /**
       * 方法 set()：设置钟表时间，h-小时数，m-分钟数，s-秒数。
       */
      public   void set(int h, int m, int s)   // 添加了文档注释的方法
      {   hour = h;   minute = m;   second = s;   }
      public   void show( )   // 未添加文档注释的方法
      {   System.out.println( hour +":" +minute +":" +second );   }
      /**
       * 方法 toString()：将时分秒数据转成字符串格式（重写从 Object 继承来的方法）。
       */
      public String toString( )   // 添加了文档注释的方法
      {   return String.format("Clock@%d:%d:%d", hour, minute, second);   }
      /**
       * 构造方法：设置钟表时间，h-小时数，m-分钟数，s-秒数。
       */
      public DocClock(int h, int m, int s)   // 添加了文档注释的方法
      {   hour = h;   minute = m;   second = s;}
}
```

例 5-25 一个定义和使用注解的钟表类 AnnoClock 示例代码（AnnoClock.java）

```
import java.lang.annotation.*;   // 导入定义注解所需的类

@Documented          // @Document 表示下面的注解 Author 可以被 javadoc 识别并提取
@interface Author {   // 定义一个注解 Author，用于生成关于作者信息的说明文档
      String value();
}

```

```
@Documented     // @Document 表示下面的注解 Info 可以被 javadoc 识别
@interface Info {   // 定义一个注解 Info，用于生成关于版本和日期的说明文档
        int version()    default 2;
        String date()    default "2018/01/01";
}

/**
 * 类的功能简介：AnnoClock 是一个钟表类，其中同时添加了文档注释和注解。
 */
@Author( "Kan" )   // 使用注解 Author 来说明类的作者信息
@Info (version = 1, date= "2018/08/20")   // 使用注解 Info 来说明类的版本和日期
public class AnnoClock {                          // 同时添加了文档注释和注解的钟表类
      ……   // 省略部分代码，参见例 5-24
      /**
       * 方法 set()：设置钟表时间，h-小时数，m-分钟数，s-秒数。
       */
      @Author( "Tom" )
      public    void set(int h, int m, int s)   // 同时添加了文档注释和注解的方法
      {   hour = h;   minute = m;   second = s;   }
      public    void show( )   // 未添加文档注释或注解的方法
      {   System.out.println( hour +":" +minute +":" +second );   }
      ……   // 省略部分代码，参见例 5-24
}
```

第 6 章 图形用户界面程序

例 6-1 一个使用框架窗口类 JFrame 编写的图形用户界面演示程序（GUITest.java）

```
import java.awt.*;            // 导入 java.awt 包中的类
import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
import javax.swing.*;      // 导入 javax.swing 包中的类

public class GUITest {                              // 主类
      public static void main(String[] args) {   // 主方法
            JFrame w = new JFrame();          // 创建框架窗口类 JFrame 的对象
            w.setTitle("图形用户界面演示程序");   // 设置窗口标题
            w.setLocation(100, 100);                  // 设置窗口位置
            w.setSize(460, 300);                        // 设置窗口尺寸
            w.setVisible(true);                         // 设置窗口为可见状态
            w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );   // 关闭窗口时退出程序
}   }
```

例 6-2 一个在窗口中绘图的 Java 演示程序（HelloWorld.java）

```
1  import java.awt.*;            // 导入 java.awt 包中的类
2  import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3  import javax.swing.*;       // 导入 javax.swing 包中的类
4
5  public class HelloWorld {                          // 主类
6      public static void main(String[] args) {   // 主方法
7          JFrame w = new JFrame();          // 创建框架窗口类 JFrame 的对象
8          w.setTitle("图形用户界面演示程序");   // 设置窗口标题
9          w.setSize(460, 300);                        // 设置窗口尺寸
10         w.setLocation(100, 100);                    // 设置窗口位置
11         w.setVisible(true);                         // 设置窗口为可见状态
12         w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );   // 关闭窗口时退出程序
13         // 以下为在窗口中绘图的 Java 代码
14         Graphics g = w.getGraphics();               // 获取窗口的绘图对象
15         Font ef = new Font("TimesRoman", Font.PLAIN, 16);   // 选择字体
16         g.setFont( ef );                                    // 设置字体
17         g.drawString("Hello, World!", 20, 80);              // 显示文字信息
18         Font cf = new Font("楷体", Font.PLAIN, 24);         // 选择字体
19         g.setFont( cf );                                    // 设置字体
20         g.drawString("你好，世界！", 20, 120);              // 显示文字信息
21         g.setColor( Color.BLACK );          // 设置填充颜色
22         g.fillRect(20, 150, 100, 100);      // 画一个实心矩形
23         g.setColor( Color.RED );            // 设置绘图颜色
24         g.drawRect(20, 150, 100, 100);   // 画一个矩形框，此处是为上面的实心矩形加框
25 }  }
```

例 6-3 一个继承框架窗口类 JFrame 并重写 paint()方法的 Java 示例程序（HelloWorld1.java）

```
1  import java.awt.*;            // 导入 java.awt 包中的类
2  import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3  import javax.swing.*;       // 导入 javax.swing 包中的类
4
5  public class HelloWorld1 {                         // 主类
6      public static void main(String[] args) {   // 主方法
7          MainWnd w = new MainWnd();   // 创建并显示主窗口对象
8          w.repaint();                                // 调用窗口的重绘方法
9  }  }
10
11 class MainWnd extends JFrame {   // 定义主窗口类：继承并扩展框架窗口类 JFrame
12     publicMainWnd() {                    // 构造方法：完成初始化窗口的功能
13         setTitle("图形用户界面演示程序");   // 设置窗口标题
14         setSize(460, 300);                          // 设置窗口尺寸
15         setLocation(100, 100);                      // 设置窗口位置
```

```
16              setVisible(true);                          // 设置窗口时显示或英寸
17              setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );  // 关闭窗口时退出程序
18          }
19
20          public void paint(Graphics g) {   // 重写绘图方法 paint()
21              super.paint( g );             // 调用超类的 paint()方法
22              Font ef = new Font("TimesRoman", Font.PLAIN, 16);  // 创建字体对象
23              g.setFont( ef );                                    // 设置字体
24              g.drawString("Hello, World!", 20, 80);              // 显示英文信息
25              Font cf = new Font("楷体", Font.PLAIN, 24);         // 选择字体
26              g.setFont( cf );                                   // 设置字体
27              g.drawString("你好，世界！", 20, 120);              // 显示中文信息
28              g.setColor( Color.BLACK );          // 设置填充颜色
29              g.fillRect(20, 150, 100, 100);      // 画一个实心矩形
30              g.setColor( Color.RED );            // 设置绘图颜色
31              g.drawRect(20, 150, 100, 100);  // 画一个矩形框，此处是为上面的实心矩形加框
32  }  }
```

例 6-4 一个在窗口中添加图形组件的 Java 示例程序（JComponentTest.java）

```
1   import java.awt.*;          // 导入 java.awt 包中的类
2   import java.awt.event.*;  // 导入 java.awt.event 包中定义的事件类
3   import javax.swing.*;      // 导入 javax.swing 包中的类
4
5   public class JComponentTest {                  // 主类
6           public static void main(String[] args) {   // 主方法
7               MainWnd w = new MainWnd();   // 创建并显示主窗口对象
8   }}
9
10  class MainWnd    extends JFrame {           // 扩展 JFrame
11          private    JButton bEN, bCN;            // 添加两个按钮字段
12          private    JLabel msg = new JLabel();   // 再添加一个标签字段
13          public    MainWnd() {                   // 构造方法
14              // 初始化窗口
15              setTitle( "图形界面演示程序" );
16              setSize(460, 300);setLocation(100, 100);
17              setVisible(true);
18              setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
19              //初始化图形组件
20              bEN = new JButton( "English" );        // 创建按钮对象
21              bCN = new JButton( "中文" );
22              msg.setOpaque(true);                    // 设置标签背景是否透明：不透明
23              msg.setBackground(Color.YELLOW);  // 设置标签背景颜色：黄色
24              msg.setText( "Hello, World!");          // 在标签上显示文本信息
25              //将组件添加到窗口的内容面板上
```

```
26              Container cp = getContentPane();   // 获得窗口的内容面板（容器）
27              cp.setLayout( null );                  // 设置容器的布局形式：null-手工布局
28              cp.add( bEN );   cp.add( bCN );   cp.add( msg );   // 在容器中添加组件
29              bEN.setBounds(10, 10, 200, 50);    // 手工设置各组件的位置和尺寸
30              bCN.setBounds(10, 70, 200, 50);
31              msg.setBounds(10, 150, 200, 80);
32              cp.validate();                          // 检查并布局容器里的组件
33  }  }
```

例 6-5 一个使用流式布局 FlowLayout 的 Java 示例程序（LayoutTest.java）

```
1   import java.awt.*;          // 导入 java.awt 包中的类
2   import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3   import javax.swing.*;      // 导入 javax.swing 包中的类
4
5   public class LayoutTest {                       // 主类
6       public static void main(String[] args) {   // 主方法
7           JButton btn[] = {   // 创建一个按钮对象数组，包含 9 个按钮
8               new JButton("Button1"), new JButton("Button2"), new JButton("Button3"),
9               new JButton("Button4"), new JButton("Button5"), new JButton("Button6"),
10              new JButton("Button7"), new JButton("Button8"), new JButton("Button9")
11          };
12          JFrame w = new JFrame();                        // 创建程序窗口
13          w.setSize(500, 200);   w.setLocation(100, 100);   // 初始化窗口
14          w.setVisible(true);
15          w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
16          // 下面演示：设置内容面板的布局策略，然后添加按钮并自动布局
17          w.setTitle( "流式布局 FlowLayout" );   // 设置窗口标题
18          Container cp = w.getContentPane();     // 获得窗口 w 的内容面板
19          FlowLayout fl = new FlowLayout();      // 创建流式布局对象
20          fl.setAlignment(FlowLayout.LEFT);      // 设置流式布局为左对齐
21          cp.setLayout( fl );   // 将内容面板（容器）的布局策略设为流式布局
22          for (int n = 0; n < btn.length; n++)   // 在内容面板中放入 9 个按钮组件
23              cp.add( btn[n] );
24          cp.validate();                          // 检查并自动布局容器里的组件
25  }  }
```

例 6-6 一个 HelloWorld 程序例子（JButtonTest.java）

```
1  import java.awt.*;            // 导入 java.awt 包中的类
2  import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3  import javax.swing.*;      // 导入 javax.swing 包中的类
4
5  public class JButtonTest {                    // 主类
6      public static void main(String[] args) {   // 主方法
7          MainWnd w = new MainWnd();   // 创建并显示程序主窗口
8  }  }
9
10 class MainWnd   extends JFrame {          // 扩展 JFrame
11     private   JButton bEN, bCN;            // 添加两个功能按钮
12     private   JLabel msg = new JLabel();   // 添加一个信息显示区标签
13     public MainWnd() {                     // 构造方法
14         // 初始化窗口
15         setTitle( "图形用户界面演示程序" );
16         setSize(460, 300);   setLocation(100, 100);
17         setVisible(true);
18         setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
19         // 初始化功能按钮，并将功能按钮放入一个子面板（JPanel）
20         bEN = new JButton( "English Button" );   // 创建英文按钮
21         bCN = new JButton( "中文按钮" );         // 创建中文按钮
22         JPanel bp = new JPanel();           // 创建放置按钮的子面板（默认流式布局）
23         bp.add( bEN );   bp.add( bCN );   // 将两个按钮放入按钮面板
24         // 初始化信息显示区标签
25         msg.setOpaque(true);   // 如需设置组件的背景色，首先需将背景设为不透明
26         msg.setBackground(Color.WHITE);                  // 设置标签的背景色
27         msg.setText( "Information area(信息显示区)");   // 设置标签里的文本内容
28         // 将按钮面板和信息标签放入窗口的内容面板
29         Container cp = getContentPane();      // 获得窗口的内容面板（默认边框布局）
30         cp.add(bp, BorderLayout.NORTH);       // 将按钮面板放在内容面板的上部
31         cp.add(msg, BorderLayout.CENTER);   // 将信息标签放在内容面板的中间
32         cp.validate();                         // 检查并自动布局容器里的组件
33 }  }
```

例 6-7 在例 6-6 代码基础上添加事件响应机制的 HelloWorld 程序例子（JButtonTest.java）

```
1~9   ……  // 第 1~9 行与例 6-6 相同，此处省略
10    class MainWnd   extends JFrame {          // 扩展 JFrame
11          private   JButton bEN, bCN;          // 添加两个功能按钮
12          private   JLabel msg = new JLabel();  // 添加一个信息显示区标签
13          public MainWnd() {                    // 构造方法
14~32             ……  // 第 14~32 行与例 6-6 相同，此处省略
33
34              // 新添加代码：为“中文按钮”注册一个处理 ActionEvent 事件的监听器对象
35              bCN.addActionListener( new BcnClicked() );
36    }  }
37
38    // 新添加代码：定义一个处理 ActionEvent 事件的监听器类 BcnClicked
39    class BcnClicked implements ActionListener {          // 需实现规定的接口 ActionListener
40          public void actionPerformed(ActionEvent e) {   // 实现接口的抽象方法 actionPerformed()
41              msg.setText( "你好，中国！");              // 在信息标签 msg 中显示反馈信息
42    }  }
```

例 6-8 一个文本字段类 JTextField 的 Java 演示程序（JTextFieldTest.java）

```
1     import java.awt.*;          // 导入 java.awt 包中的类
2     import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3     import javax.swing.*;      // 导入 javax.swing 包中的类
4
5     public class JTextFieldTest {                    // 主类
6           public static void main(String[] args) {   // 主方法
7               MainWnd w = new MainWnd();   // 创建并显示程序主窗口
8     }  }
9
10    class MainWnd   extends JFrame {                  // 扩展 JFrame
11          JTextField tf = new JTextField();           // 添加一个单行文本编辑框
12          JLabel msg = new JLabel( "Hello, World!");   // 添加一个显示信息的标签
13          public MainWnd() {                          // 构造方法
14              setTitle( "图形界面演示程序" );         // 初始化窗口
15              setSize(300, 200);   setLocation(100, 100);   setVisible(true);
16              setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17              // 设置单行文本编辑框 tf
18              tf.setBackground(Color.YELLOW);              // 设置背景色
19              tf.addActionListener( new ActionListener() {   // 添加 ActionEvent 事件监听器
20                    public void actionPerformed(ActionEvent e) {
21                          msg.setText( "Hello, " +tf.getText() );
22                    }
23              } );
24              // 在窗口的内容面板上添加组件 tf 和 msg
```

```
25              Container cp = getContentPane();   // 获得窗口的内容面板（默认边框布局）
26              cp.add( tf, BorderLayout.NORTH );   cp.add( msg, BorderLayout.CENTER );
27              cp.validate();                      // 检查并自动布局容器里的组件
28  }   }
```

例 6-9 一个文本区域类 JTextArea 的 Java 演示程序（JTextAreaTest.java）

```
1~9  ……   // 第 1~9 行与例 6-8 相同，此处省略。注：将主类名改为 JTextAreaTest
10  class MainWnd    extends JFrame {                 // 扩展 JFrame
11          JTextArea ta = new JTextArea(2, 10);       // 添加一个 2 行 10 列的文本编辑框
12          JLabel msg = new JLabel();                 // 添加一个显示信息的标签
13          JButton b = new JButton( "显示文本" );   // 添加一个按钮
14          public MainWnd() {                         // 构造方法
15                  setTitle( "图形界面演示程序" );    // 初始化窗口
16                  setSize(300, 200);    setLocation(100, 100);    setVisible(true);
17                  setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
18                  // 设置多行文本编辑框 ta
19                  ta.setBackground(Color.YELLOW);                  // 设置多行文本编辑框的背景色
20                  JScrollPane taScroller = new JScrollPane(ta);   // 将编辑框放入一个滚动面板
21                  b.addActionListener( new ActionListener() {   // 为按钮添加 ActionEvent 事件监听器
22                          public void actionPerformed(ActionEvent e) {
23                                  msg.setText( ta.getText() );      // 取出编辑框里的内容，并显示到标签中
24                          }
25                  } );
26                  // 在窗口的内容面板上添加滚动面板 taScroller、标签 msg 和按钮 b
27                  Container cp = getContentPane();   // 获得窗口的内容面板（默认边框布局）
28                  cp.add(taScroller, BorderLayout.NORTH );
29                  cp.add( msg, BorderLayout.CENTER );    cp.add( b, BorderLayout.SOUTH );
30                  cp.validate();                         // 检查并自动布局容器里的组件
31  }   }
```

例 6-10 一个单选按钮类 JRadioButton 的 Java 演示程序（JRadioButtonTest.java）

```
1~9   ……  // 第 1~9 行与 6.4.3 小节例 6-8 相同，此处省略。注：将主类名改为 JRadioButtonTest
10    class MainWnd   extends JFrame {                              // 扩展 JFrame
11        JRadioButton cbEN = new JRadioButton("英文", true);   // 单选按钮：英文
12        JRadioButton cbCN = new JRadioButton("中文");          // 单选按钮：中文
13        JRadioButton cbSH = new JRadioButton("上海话");       // 单选按钮：上海话
14        JLabel hello = new JLabel("Hello, World!", SwingConstants.CENTER);   // 信息标签
15
16        public MainWnd() {                              // 构造方法
17            setTitle( "图形界面演示程序" );   // 初始化窗口
18            setSize(300, 200);   setLocation(100, 100);   setVisible(true);
19            setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
20            // 新建一个按钮面板，放入 3 个单选按钮
21            JPanel bp = new JPanel( );   // 创建子面板（默认流式布局）
22            bp.add(cbEN);   bp.add(cbCN);   bp.add(cbSH);   // 添加单选按钮组件
23            // 将 3 个互斥的单选按钮合成一组，同时只会有一个被选中
24            ButtonGroup group = new ButtonGroup();   // 创建组对象
25            group.add(cbEN);   group.add(cbCN);   group.add(cbSH);   // 加入组中
26            // 在主窗口的内容面板中放入按钮面板 bp 和标签 hello
27            Container cp = getContentPane();        // 获得窗口的内容面板（默认边框布局）
28            cp.add( bp, BorderLayout.NORTH );       // 添加按钮面板 bp
29            cp.add( hello, BorderLayout.CENTER );   // 添加信息标签 hello
30            cp.validate();                                  // 检查并自动布局容器里的组件
31            // 处理 ItemEvent 事件的监听器：根据单选按钮状态来显示对应的信息
32            ItemListener il = new ItemListener() {                    // 匿名类
33                public void itemStateChanged(ItemEvent e) {   // 处理 ItemEvent 事件的方法
34                    String msg = null;
35                    if ( cbEN.isSelected() )    msg = "Hello, World!";
36                    else if ( cbCN.isSelected() )    msg = "你好，世界！";
37                    else if ( cbSH.isSelected() )    msg = "侬好，世界！";
38                    hello.setText( msg );
39            }   };
40            // 三个单选按钮对象共用同一个监听器对象 il
41            cbEN.addItemListener(il);   cbCN.addItemListener(il);   cbSH.addItemListener(il);
42    }   }
```

例 6-12 一个下拉列表类 JComboBox<E>的 Java 演示程序（JComboBoxTest.java）

```
1~9   ……  // 第 1~9 行与 6.4.3 小节例 6-8 相同，此处省略。注：将主类名改为 JComboBoxTest
10    class MainWnd   extends JFrame {   // 扩展 JFrame
11        JComboBox<String> list;          // 字符串型下拉列表
12        String listItems[] = { "英文", "中文", "上海话", "广东话", "闽南话" };   // 列表选项
13        JLabel info = new JLabel("", SwingConstants.CENTER);                    // 信息标签
14
15        public MainWnd() {                          // 构造方法
16            setTitle( "图形界面演示程序" );   // 初始化窗口
17            setSize(300, 200);   setLocation(100, 100);   setVisible(true);
18            setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
19            // 设置下拉列表 list
20            list = new JComboBox<String>( listItems );   // 创建下拉列表并初始化选项
21            list.setMaximumRowCount( 3 );                // 设置下拉行数
22            list.setSelectedIndex( 1 );   // 设置初始选中的列表选项（编号从 0 开始）
23            // 在窗口的内容面板上添加组件
24            Container cp = getContentPane();      // 获得窗口的内容面板（默认边框布局）
25            cp.add( list, BorderLayout.NORTH );   // 将下拉列表添加到主窗口
26            cp.add( info, BorderLayout.SOUTH );   // 将信息显示标签添加到主窗口
27            cp.validate();                          // 检查并自动布局容器里的组件
28            // 处理 ItemEvent 事件的监听器：根据下拉列表选中的选项来显示对应的信息
29            ItemListener il = new ItemListener() {                  // 匿名类
30                public void itemStateChanged(ItemEvent e) {   // 处理 ItemEvent 事件的方法
31                    JComboBox cb = (JComboBox)e.getSource();   // 获取事件源
32                    String item = (String)cb.getSelectedItem();     // 获取被选中的选项
33                    info.setText( item +"  被选中！ " );
34            }   };
35            list.addItemListener(il);   // 为下拉列表添加 ItemListener 监听器
36    }   }
```

例 6-18 一个弹出消息对话框的 Java 演示程序（JOptionPaneTest.java）

```
1  import java.awt.*;            // 导入 java.awt 包中的类
2  import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3  import javax.swing.*;       // 导入 javax.swing 包中的类
4
5  public class JOptionPaneTest {                    // 测试类
6      public static void main(String[] args) {      // 主方法
7          JFrame w = new JFrame();                  // 创建并显示主窗口对象
8          w.setTitle( "图形界面演示程序" );   // 初始化主窗口
9          w.setSize(300, 200);    w.setLocation(100, 100);    w.setVisible(true);
10         w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11         // 主窗口内容面板：按钮 +标签
12         Container cp = w.getContentPane();   // 获得窗口的内容面板（默认边框布局）
13         JButton btn = new JButton("测试对话框");   // 按钮：单击按钮弹出对话框
14         JLabel info = new JLabel("Hello!");              // 标签：显示对话框返回的结果
15         cp.add( btn, BorderLayout.NORTH );    cp.add( info, BorderLayout.CENTER );
16         cp.validate();                                   // 检查并自动布局内容面板里的组件
17         // 为按钮添加事件监听器：用户点击按钮，弹出对话框
18         btn.addActionListener( new ActionListener() {          // 匿名类
19             public void actionPerformed(ActionEvent e) {   // 弹出消息对话框
20                 JOptionPane.showMessageDialog(w, "欢迎进入课程学习",
21                     "Java 语言程序设计", JOptionPane.INFORMATION_MESSAGE);
22         }} );
23 }  }
```

例 6-19 一个文件选择类 JFileChooser 的 Java 演示程序（JFileChooserTest.java）

```
1  import javax.swing.*;   // 导入 javax.swing 包中的类
2  import java.io.File;       // 导入 java.io 包中定义的文件类 File
3
4  public class JFileChooserTest {                    // 测试类
5      public static void main(String[] args) {       // 主方法
6          JFileChooser fc = new JFileChooser();   // 创建一个文件选择类的对象
7          fc.setCurrentDirectory( new File("D:/我的 Java 语言/Example/Chapter1/src") );
8          fc.showOpenDialog(null);                   // 弹出打开文件对话框
9          File f = fc.getSelectedFile();             // 返回所选择的文件
10         JOptionPane.showMessageDialog(null, "您选择的文件是：" +f.getName());
11 }  }
```

第 7 章 输入输出流

例 7-1 一个使用 System.in 对象进行键盘输入的 Java 演示程序（JSystemInTest.java）

```
1  import java.io.*;   // 导入 java.io 包中的类
2  public class JSystemInTest {                    // 测试类
3      public static void main(String[] args) {    // 主方法
4          byte bbuf[] = new byte[20];          // 定义字节数组保存键盘输入的字节流
5          // 按字节流方式读取键盘输入的数据
6          try {   // 必须处理输入过程中可能抛出的勾选异常 IOException
7              int len = System.in.read(bbuf);   // 此处可能会抛出 IOException 异常
8              for (int n = 0; n <len; n++) {
9                  int x = Byte.toUnsignedInt( bbuf[n] );        // 将字节流先转成无符号整数
10                 String hexString = Integer.toHexString(x);   // 再转成十六进制字符串
11                 System.out.print(hexString +" ");            // 依次显示输入的字节流
12             }
13             System.out.println();
14         }
15         catch(IOException e)   // 处理勾选异常 IOException
16         {   System.out.println( e.getMessage() );   }
17 }  }
```

例 7-2 一个使用输入流包装类 InputStreamReader 的 Java 演示程序（InputStreamReaderTest.java）

```
1  import java.io.*;   // 导入 java.io 包中的类
2  public class InputStreamReaderTest {            // 测试类
3      public static void main(String[] args) {    // 主方法
4          char cbuf[] = new char[20];          // 定义字符数组保存键盘输入的字符流
5          try {   // 必须处理勾选异常 IOException
6              InputStreamReader inChar = new InputStreamReader(System.in);   // 包装
7              int len = inChar.read( cbuf );   // 此处可能会抛出 IOException 异常
8              for (int n = 0; n <len; n++) {     // 显示所输入的字符数据
9                  System.out.print(cbuf[n] +" ");   // 字符之间用空格隔开
10             }
11             System.out.println();
12             inChar.close();   // 关闭输出流对象 inChar
13         }
14         catch(IOException e)   // 处理勾选异常 IOException
15         {   System.out.println( e.getMessage() );   }
16 }  }
```

例 7-3 一个使用扫描器类 Scanner 进行格式化输入的 Java 演示程序（JScannerTest.java）

```
1  import java.io.*;              // 导入 java.io 包中的类
2  import java.util.Scanner;   // 导入 java.util 包中定义的扫描器类 Scanner
3
4  public class JScannerTest {                    // 测试类
5        public static void main(String[] args) {   // 主方法
6              Scanner sc = null;                  // 定义一个扫描器引用变量
7              System.out.print( "从键盘输入数据：" );
8              try {   // 输入时可能会抛出异常，使用扫描器类建议按此代码框架编写程序
9                    sc = new Scanner( System.in );   // 将键盘对象 System.in 包装成扫描器对象
10                   String str = sc.next();             // 读取下一个字符串
11                   int x = sc.nextInt();                // 读取下一个 int 型整数
12                   System.out.println( "输入结果为：" +str +", " +x );   // 显示输入结果
13             }
14             finally
15             {   if (sc != null)   sc.close();   }   // 关闭扫描器
16 }  }
```

例 7-4 一个使用 System.out 对象进行格式化输出的 Java 演示程序（JPrintStreamTest.java）

```
1  import java.io.*;   // 导入 java.io 包中的类
2  public class JPrintStreamTest {                 // 测试类
3        public static void main(String[] args) {   // 主方法
4              int x = 10;                          // 先定义几个演示数据
5              double y = 15.8;
6              String str = "abcd";
7              // 下面演示格式化输出方法
8              System.out.println(x);     // 输出一个 int 型整数
9              System.out.println(y);     // 输出一个 double 型实数
10             System.out.println(str);   // 输出一个 String 字符串
11             // 下面演示同时格式化输出多个数据的方法
12             System.out.println("x=" +x +", y=" +y +"str= " +str);     // 输出一个字符串表达式
13             System.out.format("x=%d, y=%f, str=%s\n", x, y, str);   // 格式化输出多个数据项
14             System.out.format("x=%h\n", x);                          // 显示十六进制整数
15             System.out.printf("x=%d, y=%f, str=%s\n", x, y, str);     // 类似于 C 语言的 printf 函数
16 }  }
```

例 7-5 一个完整的文本文件输入输出演示程序（JFileWRTest.java）

```
1  import java.io.*;                // 导入 java.io 包中的类
2  public class JFileWRTest {   // 测试类：演示文件流类 FileWriter 和 FileReader 的用法
3      public static void main(String[] args) {   // 主方法
4          fwrite("d:/fwr.txt");   // 调用下面的方法 fwrite()：创建并输出一个文本文件
5          fread("d:/fwr.txt");   // 再调用下面的方法 fread()：输入并显示文本文件中的内容
6      }
7      static void fwrite(String fileName) {   // 创建并输出文本文件的方法
8          try {   // 必须处理勾选异常 IOException
9              FileWriter fw = new FileWriter(fileName);          // 创建字符型文件输出流
10             fw.write("中国农业大学作为教育部直属高校，\r\n");   // 向文件写入数据
11             fw.write("其历史起自于 1905 年成立的京师大学堂农科大学。\r\n");
12             fw.write("1949 年 9 月，由三所大学下属的农学院合并而成……");
13             fw.close();   // 关闭文件输出流
14             System.out.println(fileName +"输出成功。");
15             System.out.println();
16         }
17         catch(IOException e)   // 处理 IOException 异常（勾选异常）
18         {   System.out.println( e.getMessage() );   }
19     }
20     static void fread(String fileName) {   // 输入文本文件并显示到显示器的方法
21         try {   // 必须处理勾选异常 IOException
22             FileReader fr = new FileReader(fileName);   // 创建字符型文件输入流
23             int c;
24             while ( (c = fr.read()) != -1 ) {              // 从文件读取字符，读完为止
25                 System.out.print( (char)c );//显示所读出的字符 c
26             }
27             fr.close();   // 关闭文件输入流
28             System.out.println();
29             System.out.println(fileName +"输入成功。");
30         }
31         catch(IOExceptione)   // 处理 IOException 异常（勾选异常）
32         {   System.out.println( e.getMessage() );   }
33 }  }
```

例 7-6 一个使用缓冲区进行文本文件 IO 的 Java 演示程序（JBufferedWRTest.java）

```
1  import java.io.*;  // 导入 java.io 包中的类
2  public class JBufferedWRTest {                  // 测试类
3      public static void main(String[] args) {  // 主方法
4          fwrite("d:/bwr.txt");  // 调用下面的方法 fwrite()：创建并输出一个文本文件
5          fread("d:/bwr.txt");  // 再调用下面的方法 fread()：输入并显示文本文件中的内容
6      }
7      static void fwrite(String fileName) {  // 创建并输出文本文件的方法
8          try {  // 必须处理勾选异常 IOException
9              // 将字符型文件输出流对象包装成带缓冲区的字符型文件输出流对象
10             BufferedWriter bw = new BufferedWriter( new FileWriter(fileName) );
11             bw.write("中国农业大学作为教育部直属高校，");
12             bw.newLine();  // BufferedWriter 增加了换行方法 newLine()
13             bw.write("其历史起自于 1905 年成立的京师大学堂农科大学。");
14             bw.newLine();
15             bw.write("1949 年 9 月，由三所大学下属的农学院合并而成……");
16             bw.close();  // 关闭文件输出流
17             System.out.println(fileName +"输出成功。");
18             System.out.println();
19         }
20         catch(IOException e)  // 处理 IOException 异常（勾选异常）
21         {  System.out.println( e.getMessage() );  }
22     }
23     static void fread(String fileName) {  // 输入文本文件并显示到显示器的方法
24         try {  // 必须处理勾选异常 IOException
25             // 将字符型文件输入流对象包装成带缓冲区的字符型文件输入流对象
26             BufferedReader br = new BufferedReader( new FileReader(fileName) );
27             String s;
28             while ( (s = br.readLine()) != null ) {  // BufferedReader 增加了读一行的方法
29                 System.out.print( s );          // 所读出的字符串中去掉了回车和换行符
30                 System.out.println();
31             }
32             br.close();  // 关闭文件输入流
33             System.out.println(fileName +"输入成功。");
34         }
35         catch(IOException e)  // 处理 IOException 异常（勾选异常）
36         {  System.out.println( e.getMessage() );  }
37 }  }
```

例 7-7 一个对文本文件进行格式化输入输出的 Java 演示程序（JFormatWSTest.java）

```
1  import java.io.*;              // 导入 java.io 包中的类
2  import java.util.Scanner;   // 导入 java.util 包中定义的扫描器类 Scanner
3  public class JFormatWSTest {                    // 测试类
4      public static void main(String[] args) {   // 主方法
5          fprint("d:/pws.txt");   // 调用下面的方法 fprint()：创建并输出一个文本文件
6          fscan("d:/pws.txt");   // 再调用下面的方法 fscan()：输入并显示文本文件中的内容
7      }
8      static void fprint(String fileName) {   // 创建并格式化输出文本文件的方法
9          try { // 必须处理勾选异常 IOException
10             PrintWriter pw = new PrintWriter( fileName );   // 创建字符型文件打印流
11             int x = 10;   double y = 15.8;   String str = "abcd";
12             pw.print(x +" ");   // 输出一个 int 型整数
13             pw.print(y +" ");   // 输出一个 double 型实数
14             pw.println(str);    // 输出一个 String 字符串
15             pw.format("%d %f %s", x, y, str);   // 格式化输出多个数据项
16             pw.println();
17             pw.close();   // 关闭文件打印流
18             System.out.println(fileName +"输出成功。");   System.out.println();
19         }
20         catch(IOException e)   // 处理 IOException 异常（勾选异常）
21         {   System.out.println( e.getMessage() );   }
22     }
23     static void fscan(String fileName) {   // 格式化输入文本文件并显示到显示器的方法
24         Scanner sc = null;                  // 定义一个扫描器引用变量
25         try {   // 必须处理勾选异常 IOException
26             sc = new Scanner( new File(fileName) );   // 创建文件扫描器
27             while (sc.hasNext()) {                  // 检查扫描器中是否还有可输入的数据
28                 int x = sc.nextInt();              // 读取下一个 int 型整数
29                 double y = sc.nextDouble();   // 读取下一个 double 型实数
30                 String str = sc.next();             // 读取下一个字符串
31                 System.out.println( x +", " +y +", " +str );   // 显示输入结果
32             }
33             sc.close();   // 关闭文件扫描器
34             System.out.println(fileName +"输入成功。");
35         }
36         catch(IOException e)   // 处理 IOException 异常（勾选异常）
37         {   System.out.println( e.getMessage() );   }
38 }  }
```

例 7-8 一个简单数据序列化及二进制文件 IO 的 Java 演示程序（JObjectIO.java）

```
1  import java.io.*;  // 导入 java.io 包中的类
2  public class JObjectIO {                          // 测试类
3      public static void main(String[] args) {  // 主方法
4          fwrite("d:/sim-io.dat");  // 调用下面的方法 fwrite()输出一个二进制文件
5          fread("d:/sim-io.dat");  // 调用下面的方法 fread()输入并显示二进制文件的内容
6      }
7      static   void fwrite(String fileName) {  // 序列化并输出二进制文件的方法
8          try {  // 必须处理勾选异常 IOException
9              // 先创建字节型文件输出流对象，再包装成带序列化功能的输出流对象
10             FileOutputStream fos = new FileOutputStream(fileName);
11             ObjectOutputStream oos = new ObjectOutputStream( fos );
12             int x = 10;   double y = 15.8;   String str = "abcd 中国";  // 简单数据
13             oos.writeInt(x);          // 序列化并输出一个 int 型整数（4 字节）
14             oos.writeDouble(y);  // 序列化并输出一个 double 型实数（8 字节）
15             oos.writeUTF(str);       // 序列化并输出一个字符串（UTF-16 被转成 UTF-8）
16             oos.close();              // 关闭输出流
17             System.out.println(fileName +"输出成功。");  System.out.println();
18         }
19         catch(IOException e)  // 处理 IOException 异常（勾选异常）
20         {  System.out.println( e.getMessage() );  }
21     }
22     static void fread(String fileName) {  // 输入二进制文件并反序列化的方法
23         try {  // 必须处理勾选异常 IOException
24             // 先创建字节型文件输入流对象，再包装成带反序列化功能的输入流对象
25             FileInputStream fis = new FileInputStream(fileName);
26             ObjectInputStream ois = new ObjectInputStream( fis );
27             int x = ois.readInt();              // 输入并反序列化一个 int 型整数（4 字节）
28             double y = ois.readDouble();  // 输入并反序列化一个 double 型实数（8 字节）
29             String str = ois.readUTF();      // 输入并反序列化一个字符串（UTF-8 转 UTF-16）
30             System.out.println( x +", " +y +", " +str );  // 显示输入结果
31             ois.close();  // 关闭数据输入流
32             System.out.println(fileName +"输入成功。");
33         }
34         catch(IOException e)  // 处理 IOException 异常（勾选异常）
35         {  System.out.println( e.getMessage() );  }
36 }  }
```

例 7-9 一个对钟表对象进行序列化和反序列化的 Java 演示程序（JObjectIO.java）

```
1  import java.io.*;  // 导入 java.io 包中的类
2  public class JObjectIO {                          // 测试类
3      public static void main(String[] args) {   // 主方法
4          fwrite("d:/obj-io.dat");   // 调用下面的方法 fwrite()输出一个二进制文件
5          fread("d:/obj-io.dat");   // 调用下面的方法 fread()输入并显示二进制文件的内容
6      }
7      static   void fwrite(String fileName) {   // 序列化并输出二进制文件的方法
8          try {   // 必须处理勾选异常 IOException
9              // 先创建字节型文件输出流对象，再包装成带序列化功能的输出流对象
10             FileOutputStream fos = new FileOutputStream(fileName);
11             ObjectOutputStream oos = new ObjectOutputStream( fos );
12             Clock c1 = new Clock(8, 30, 15);   // 创建两个将被序列化的钟表对象
13             Clock c2 = new Clock(10, 30, 15);
14             oos.writeObject(c1);   oos.writeObject(c2);   // 序列化并输出对象
15             oos.close();   // 关闭输出流
16             System.out.println(fileName +"输出成功。");   System.out.println();
17         }
18         catch(IOException e)   // 处理 IOException 异常（勾选异常）
19         {   System.out.println( e.getMessage() );   }
20     }
21     static void fread(String fileName) {   // 输入二进制文件并反序列化的方法
22         try {   // 必须处理勾选异常 IOException
23             // 先创建字节型文件输入流对象，再包装成带反序列化功能的输入流对象
24             FileInputStream fis = new FileInputStream(fileName);
25             ObjectInputStream ois = new ObjectInputStream( fis );
26             Clock c1 = (Clock)ois.readObject();   // 输入并反序列化第 1 个钟表对象
27             Clock c2 = (Clock)ois.readObject();   // 输入并反序列化第 2 个钟表对象
28             c1.show();   c2.show();   // 显示反序列化得到的钟表对象时间
29             ois.close();                    // 关闭数据输入流
30             System.out.println(fileName +"输入成功。");
31         }
32         catch(IOException e)   // 处理 IOException 异常（勾选异常）
33         {   System.out.println( e.getMessage() );   }
34         catch(ClassNotFoundException e)   // 处理勾选异常 ClassNotFoundException
35         {   System.out.println( e.getMessage() );   }
36 }  }
37
38 class Clockimplements Serializable {   // 定义钟表类时激活序列化功能
39     // 必须添加一个 long 型字段 serialVersionUID，为类指定一个序列化编号
40     private static final long serialVersionUID = 2018L;   // 本例将编号指定为 2018
41     private int hour, minute, second;   // 字段：时分秒
42     public   void show()                    // 显示时间
43     {   System.out.println( hour +":" +minute +":" +second );   }
```

```
44        public    Clock(int h, int m, int s)    // 构造方法
45        {    hour = h;    minute = m;    second = s;    }
46  }
```

例 7-11 一个使用字符串类 String 进行分词的 Java 演示程序（JTextSplit.java）

```
1   public class JTextSplit {                          // 测试类
2         public static void main(String[] args) {    // 主方法
3               String str= "I am in Beijing";         // 待分词的字符串
4               String words[];                        // 字符串数组，用于保存分词结果
5               // 对存储在 str 中的字符串"I am in Beijing"进行分词
6               words = str.split(" ");    // 将空格作为分隔符进行分词
7               System.out.println("\"" +str +"\"的分词结果如下：" );
8               for (String w: words)    // 显示分词结果
9               {    System.out.print("[" +w +"] ");    }
10              System.out.println();    System.out.println();    // 换行
11              // 再对字符串"one,two,three,four,five"进行分词
12              str = "one,two,three,four,five";    // 待分词的字符串
13              words = str.split(",");                // 将逗号作为分隔符进行分词
14              System.out.println("\"" +str +"\"的分词结果如下：" );
15              for (String w: words)                  // 显示分词结果
16              {    System.out.print("[" +w +"] ");    }
17              System.out.println();
18  }   }
```

例 7-16 一个图标类 ImageIcon 的 Java 演示程序（JImageIconTest.java）

```
1  import java.awt.*;            // 导入 java.awt 包中的类
2  import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3  import javax.swing.*;       // 导入 javax.swing 包中的类
4
5  public class JImageIconTest {                    // 测试类
6      public static void main(String[] args) {   // 主方法
7          JFrame w = new JFrame();              // 创建框架窗口类 JFrame 的对象
8          w.setTitle("图像演示程序");          // 初始化窗口
9          w.setSize(420, 320);    w.setLocation(100, 100);    w.setVisible(true);
10         w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11         // 在窗口内容面板里添加一个显示图像的标签组件
12         Container cp = w.getContentPane();   // 获得窗口的内容面板（默认边框布局）
13         JLabel box = new JLabel();
14         cp.add(box, BorderLayout.CENTER);     // 将标签放在内容面板的中间
15         cp.validate();                                  // 检查并自动布局容器里的组件
16         // 从图像文件加载图像，创建一个图标对象
17         ImageIcon ii = new ImageIcon("d:/1.jpg");
18         box.setIcon(ii);   // 在标签组件中显示图像
19 }  }
```

例 7-17 一个带缓存图像类 BufferedImage 的 Java 演示程序（JBufferedImageTest.java）

```
1  import java.awt.*;            // 导入 java.awt 包中的类
2  import java.awt.event.*;   // 导入 java.awt.event 包中定义的事件类
3  import javax.swing.*;       // 导入 javax.swing 包中的类
4  import java.io.*;              // 导入 java.io 包中的类
5  import java.awt.image.BufferedImage;   // 导入 java.awt.image 包中的类 BufferedImage
6  import javax.imageio.ImageIO;             // 导入 javax.imageio 包中的类 ImageIO
7
8  public class JBufferedImageTest {                // 测试类
9      public static void main(String[] args) {   // 主方法
10         JFrame w = new JFrame();          // 创建框架窗口类 JFrame 的对象
11         w.setTitle("图像演示程序");       // 初始化窗口
12         w.setSize(420, 320);    w.setLocation(100, 100);    w.setVisible(true);
13         w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
14         // 从文件加载图像，创建一个带缓存的图像对象
15         BufferedImage bi = null;
16         try {   // 必须处理勾选异常 IOException
17             File fin = new File("d:/1.jpg");
18             bi = ImageIO.read(fin);   // 图像输入输出类 ImageIO
19         }
20         catch(IOException e)   // 处理 IOException 异常（勾选异常）
21         {   System.out.println( e.getMessage() );    return;    }
```

```
22              // 修改图像：将图像中各像素的颜色值设为其补色
23              Color c1, c2;
24              for (int y = 0; y < bi.getHeight(); y++){              // 行循环
25                    for (int x = 0; x < bi.getWidth(); x++) {     // 列循环
26                          c1 = new Color( bi.getRGB(x, y) );   // 取出颜色值
27                          c2 = new Color( 255-c1.getRed(), 255-c1.getGreen(), 255-c1.getBlue() );
28                          bi.setRGB(x, y, c2.getRGB());          // 设为补色 c2
29              }}
30              // 在窗口内容面板里添加一个显示图像的画布组件
31              Container cp = w.getContentPane();   // 获得窗口的内容面板（默认边框布局）
32              MyCanvas cv = new MyCanvas(bi);     // 创建用于显示图像的画布对象
33              cp.add(cv, BorderLayout.CENTER);     // 将画布放在内容面板的中间
34              cp.validate();                                  // 检查并自动布局容器里的组件
35              // 保存图像：将修改后的图像保存到一个新的图像文件
36              try {   // 必须处理勾选异常 IOException
37                    File fout = new File("d:/1-new.jpg");
38                    ImageIO.write(bi, "jpg", fout);   // 图像输入输出类 ImageIO
39              }
40              catch(IOException e){   System.out.println( e.getMessage() );   }
41  }   }
42
43  class MyCanvas extends Canvas {   // 定义一个新的画布类，重写 paint()方法
44        private    BufferedImage bi = null;
45        public MyCanvas(BufferedImage i)   // 构造方法
46        {   bi = i;   }
47        public void paint(Graphics g)          // 重写 paint()方法，显示图像
48        {   if (bi != null)g.drawImage(bi, 0, 0, null);   }
49  }
```

第 8 章 多线程并发编程

例 8-1 一个模拟音乐播放器的单线程串行 Java 演示程序（JPlayerST.java）

```
1  public class JPlayerST {                          // 主类：单线程串行程序
2      public static   void main(String[] args) {   // 主方法
3          // 下载算法：模拟网络下载，显示 5 次信息"Downloading ..."
4          int count = 1;           // 显示计数
5          while (count <= 5) {   // 循环显示 5 次信息
6              System.out.println("Downloading ..." +count);
7              count++;
8              // 每显示一次信息后让算法休眠（暂停）0.1 秒，模拟下载过程
9              try {                          // 捕获并处理可能抛出的勾选异常
10                 Thread.sleep(100);   // 可能抛出勾选异常 InterruptedException
11             }
12             catch (InterruptedException e) {   // 捕捉并处理异常
13                 System.out.println( e.getMessage() );
14                 return;   // 退出主方法，程序结束
15             }
16         }
17         play();   // 下载完成后，调用子方法 play()，模拟播放音乐
18     }
19
20     static   void play() {   // 子方法
21         // 播放算法：模拟播放音乐，显示 5 次信息"Playing ..."
22         int count = 1;           // 显示计数
23         while (count <= 5) {   // 循环显示 5 次信息
24             System.out.println("\t Playing ..." +count);
25             count++;
26             // 每显示一次信息后让算法休眠（暂停）0.1 秒。模拟播放过程
27             try {                          // 捕获并处理可能抛出的勾选异常
28                 Thread.sleep(100);   // 可能抛出勾选异常 InterruptedException
29             }
30             catch (InterruptedExceptione) {   // 捕捉并处理异常
31                 System.out.println( e.getMessage() );
32                 return;   // 退出子方法，返回主方法
33             }
34         }
35 }  }
```

例 8-2 一个模拟音乐播放器的多线程并发 Java 演示程序（JPlayerMT.java）

```
1  public class JPlayerMT {                          // 主类：多线程并发程序
2      public static  void main(String[] args) {     // 主方法
3          // 将播放算法放入单独的线程中去执行
4          PlayAlgorithm a = new PlayAlgorithm();   // 创建一个可运行的播放算法对象 a
5          Thread t = new Thread(a);          // 新建子线程 t，在 t 中运行算法对象 a
6          t.start();                         // 启动子线程 t
7          // 下面的下载算法在主线程中执行，将与上面子线程中的播放算法并发执行
8          // 下载算法：模拟网络下载，显示 5 次信息"Downloading ..."
9          int count = 1;            // 显示计数
10         while (count <= 5) {   // 循环显示 5 次信息
11             System.out.println("Downloading ..." +count);   count++;
12             // 每显示一次信息后让程序休眠（暂停）0.1 秒，模拟下载过程
13             try {                          // 捕获并处理可能抛出的勾选异常
14                 Thread.sleep(100);   // 可能抛出勾选异常 InterruptedException
15             }
16             catch (InterruptedException e) {   // 捕捉并处理异常
17                 System.out.println( e.getMessage() );
18                 return;   // 退出主方法，程序结束
19             }
20         }
21         // 当主线程和子线程都执行结束时则退出程序，进程也随之结束
22 }  }
23
24 class PlayAlgorithm implements Runnable {     // 可运行的算法类 PlayAlgorithm
25     public void run() {                          // 描述算法的方法 run
26         // 播放算法：模拟播放音乐，显示 5 次信息"Playing ..."
27         int count = 1;            // 显示计数
28         while (count <= 5) {   // 循环显示 5 次信息
29             System.out.println("\t Playing ..." +count);   count++;
30             // 每显示一次信息后让程序休眠（暂停）0.1 秒。模拟播放过程
31             try {   // 捕获并处理可能抛出的勾选异常
32                 Thread.sleep(100);   // 可能抛出勾选异常 InterruptedException
33             }
34             catch (InterruptedException e) {   // 捕捉并处理异常
35                 System.out.println( e.getMessage() );
36                 return;   // 退出子方法，返回主方法
37             }
38         }
39         // 执行完算法代码后退出 run()方法，执行该方法的线程也随之结束
40 }  }
```

例 8-5 一个单线程串行的 Java 售票服务演示程序（JTicketST.java）

```
1  public class JTicketST {                          // 主类：单线程串行的售票演示程序
2      public static void main(String[] args) {   // 主方法
3          TicketBox      tb = new TicketBox(4);          // 创建票箱，初始化有 4 张票
4          TicketWindow tw = new TicketWindow(tb);   // 创建一个售票窗口
5          // 模拟通过一个售票窗口向 5 位客户提供售票服务
6          long sTime = System.currentTimeMillis();     // 记录开始时间
7          for (int n = 1; n <= 5; n++)                    // 循环提供 5 次售票服务
8              tw.serviceAlgorithm();
9          long eTime = System.currentTimeMillis();     // 记录结束时间
10         System.out.println( "用时: " +(eTime-sTime)/1000.0 +"秒" );
11 }  }
12
13 class TicketBox {                // 描述票箱的类 TicketBox
14     private int num = 0;     // 剩余票数
15     public TicketBox(int x) {    num = x;   }   // 构造方法
16     public int get() {    return num;   }       // 读取剩余票数
17     public void set(int x) {    num = x;   }    // 设置剩余票数
18 }
19
20 class TicketWindow {                        // 提供售票服务的售票窗口类
21     private TicketBox tBox;                 // 从票箱对象 tBox 中取票
22     public TicketWindow(TicketBox p)   // 构造方法
23     {   tBox = p;   }
24     public void prepare() {   // 模拟售票前的一些准备工作，例如询问出发日期、目的地等
25         System.out.println(Thread.currentThread().getName() +": 购票前准备 ...");
26         try {
27             Thread.sleep(100);         // 休眠（暂停）0.1 秒，模拟购票前的准备工作
28         }
29         catch(InterruptedException e)   // 捕捉 sleep()方法可能抛出的异常
30         {   System.out.println( e.getMessage() );   return;   }
31     }
32     public void sale() {              // 具体的售票算法
33         int tickets = tBox.get();   // 读取剩余票数
34         if (tickets > 0) {            // 如果有票
35             tickets --;               // 售出一张票，将剩余票数减一
36             tBox.set(tickets);   // 设置票箱的剩余票数
37             System.out.println(Thread.currentThread().getName() +
38                                         ": 成功，剩余票数 " +tickets);
39         }
40         else   System.out.println(Thread.currentThread().getName() +": 无票");   // 无票
41     }
42     public void serviceAlgorithm() {   // 描述完整售票服务流程的算法
43         prepare();    // 模拟售票前的准备工作
```

```
44              sale();          // 模拟售票
45  }  }
```

例 8-6 一个多线程并发的 Java 售票服务演示程序（JTicketMT.java）

```
1   public class JTicketMT {                          // 主类：多线程并发的售票演示程序
2       public static void main(String[] args) {   // 主方法
3           TicketBox       tb = new TicketBox(4);          // 创建票箱，初始化有 4 张票
4           TicketWindow tw = new TicketWindow(tb);   // 创建售票窗口（是一个算法对象）
5           // 模拟 5 个售票窗口同时向 5 位客户提供售票服务
6           Thread t1 = new Thread(tw, "窗口 1");   // 子线程 t1~t5 执行同样的售票算法 tw
7           Thread t2 = new Thread(tw, "窗口 2");   Thread t3 = new Thread(tw, "窗口 3");
8           Thread t4 = new Thread(tw, "窗口 4");   Thread t5 = new Thread(tw, "窗口 5");
9           // 启动 5 个子线程，5 个售票窗口同时开始售票
10          long sTime = System.currentTimeMillis();   // 记录开始时间
11          t1.start();   t2.start();   t3.start();   t4.start();   t5.start();
12          // 主线程等待 5 个子线程都执行结束
13          while ( t1.getState() != Thread.State.TERMINATED ||
14                  t2.getState() != Thread.State.TERMINATED ||
15                  t3.getState() != Thread.State.TERMINATED ||
16                  t4.getState() != Thread.State.TERMINATED ||
17                  t5.getState() != Thread.State.TERMINATED )
18          { }  // 空循环，等待 5 个子线程结束
19          long eTime = System.currentTimeMillis();   // 记录结束时间
20          System.out.println( "用时: " +(eTime-sTime)/1000.0 +"秒" );
21  }  }
22
23  class TicketBox { …… }  // 描述票箱的类 TicketBox，同例 8-5，省略代码
24
25  class TicketWindow {   // 提供售票服务的售票窗口类
26  class TicketWindow   implements Runnable {   // 实现 Runnable 接口才能在线程中运行
27      ……  // 此处同例 8-5 中的代码第 21~41 行，省略代码
28      public void serviceAlgorithm() {   // 描述完整售票服务流程的算法
29      public void run() {   // 必须实现 run()方法，描述可在线程中运行的售票服务算法
```

```
30              prepare();     // 模拟售票前的准备工作
31              sale();        // 模拟售票
32  }  }
```

第 9 章 网络编程

例 9-1 一个因特网地址类 InetAddress 的 Java 演示程序（JInetAddressTest.java）

```
1   import java.net.*;   // 导入 java.net 网络包中的类
2   public class JInetAddressTest {                  // 测试类：测试因特网地址类 InetAddress 的用法
3       public static void main(String[] args) {   // 主方法
4           try {   // 处理可能出现的勾选异常 UnknownHostException
5               InetAddress local = InetAddress.getLocalHost();   // 获取本机的因特网地址对象
6               System.out.println("通过 getLocalHost()获得本机因特网地址对象: " +local );
7               System.out.println("getHostName(): " +local.getHostName() );        // 主机名
8               System.out.println("getHostAddress(): " +local.getHostAddress() );   // IP 地址
9               System.out.println();
10              // 下面演示域名与主机名、IP 地址之间的关系
11              String cauWeb = "www.cau.edu.cn";              // 中国农业大学网站的主机名
12              InetAddress cau = InetAddress.getByName(cauWeb);   // 根据主机名创建对象
13              System.out.println("根据主机名创建因特网地址对象: " +cau );
14              System.out.println("getHostName(): " +cau.getHostName() );        // 主机名
15              System.out.println("getHostAddress(): " +cau.getHostAddress() );   // IP 地址
16          }
17          catch(UnknownHostException e) { e.printStackTrace(); }   // 捕捉并处理勾选异常
18  }  }
```

例 9-2 一个统一资源定位符类 URL 的 Java 演示程序（JURLTest.java）

```
1  import java.net.*;   // 导入 java.net 网络包中的类
2  import java.io.*;    // 导入 java.io 输入输出流包中的类
3  public class JURLTest {                          // 测试类：测试统一资源定位符类 URL 的用法
4      public static void main(String [] args) {   // 主方法
5          try {   // 处理可能出现的勾选异常 MalformedURLException
6              URL url = new URL("http://www.oracle.com/technetwork/java/index.html");
7              System.out.println( "URL：" + url );
8              System.out.println( "协议：" + url.getProtocol() );
9              System.out.println( "主机：" + url.getHost() );
10             System.out.println( "端口：" + url.getPort() );
11             System.out.println( "默认端口：" + url.getDefaultPort() );
12             System.out.println( "路径：" + url.getPath() );
13         }
14         catch(MalformedURLException e) { e.printStackTrace(); }  // 捕捉并处理勾选异常
15 }  }
```

例 9-3 一个访问 Java 语言网站主页的演示程序（JWebPageTest.java）

```
1  import java.net.*;   // 导入 java.net 网络包中的类
2  import java.io.*;    // 导入 java.io 输入输出流包中的类
3  public class JWebPageTest {                      // 测试类：读取网站里的网页文件（html）
4      public static void main(String [] args) {   // 主方法
5          try {   // 处理可能出现的勾选异常 IOException
6              URL url = new URL("http://www.oracle.com/technetwork/java/index.html");
7              System.out.println("从网页读取信息: " +url);
8              InputStreamReader in = new InputStreamReader( url.openStream() );
9              char cbuf[] = new char[300];   // 只读 300 个字符
10             int len = in.read(cbuf);
11             for (int n = 0; n < len; n++)
12                 System.out.print( cbuf[n]);
13             System.out.print("......\n 以上是从网页读取的原始信息。");
14             System.out.println( "字符编码是：" +in.getEncoding() );
15             in.close();
16         }
17         catch(IOException e) { e.printStackTrace(); }  // 捕捉并处理勾选异常
18 }  }
```

例 9-4 一个访问 Web 服务里图像文件的 Java 演示程序（JWebImageTest.java）

```
1  import java.net.*;   // 导入 java.net 网络包中的类
2  import java.io.*;    // 导入 java.io 输入输出流包中的类
3  import java.awt.*;   // 以下为导入图形用户界面和图像相关的类
4  import java.awt.image.BufferedImage;
5  import javax.imageio.ImageIO;
6  import javax.swing.*;
7
8  public class JWebImageTest {                    // 测试类：加载并显示网站里的图像文件
9      public static void main(String [] args) {   // 主方法
10         try {   // 处理可能出现的勾选异常 IOException
11             String netURL = "http://www.cau.edu.cn/images/1182/culture.jpg";
12             System.out.println("从网站读取图片: " +netURL);
13             URL url = new URL(netURL);
14             BufferedImage img = ImageIO.read( url);   // 加载网络图像文件
15             // 创建框架窗口，显示加载的网络图像
16             JFrame w = new JFrame("显示网络图片");   // 创建窗口
17             w.setSize(660,360);    w.setVisible(true);
18             w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19             SwingUtilities.invokeLater( ()->{        // 在事件分发线程中绘图
20                 Graphics g = w.getGraphics();   // 获取绘图对象
21                 g.setFont( new Font("Times New Rome",0,24) );   // 设置字体
22                 g.drawString(netURL, 10, 75);                   // 显示网址
23                 g.drawImage(img, 10, 100, null);                // 显示图像
24             } );
25         }
26         catch(IOException e) { e.printStackTrace(); }   // 捕捉并处理勾选异常
27 }  }
```

第 10 章 数据库编程

例 10-1 在本地 JavaDB 数据库系统中创建教务信息数据库的 Java 示例程序（JDBCreate.java）

```
1  import java.sql.*;  // 导入 java.sql 包中数据库相关的类和接口
2  public class JDBCreate {                    // 主类：初始创建一个教务信息数据库“cau”
3      public static void main(String[] args) {  // 主方法
4          try {  // 处理数据库访问过程中可能出现的勾选异常
5              // 指定 Java DB 驱动程序的类名，然后调用 Class.forName(...)加载驱动程序
6              String dbDriver = "org.apache.derby.jdbc.EmbeddedDriver";  // Java DB 驱动
7              Class.forName(dbDriver);//加载驱动
8              System.out.println(dbDriver +" loaded.");
9              // 连接 Java DB 数据库 cau（对应文件夹 D:\\cau)。如不存在则创建
10             String dbURL = "jdbc:derby:D:\\cau; create=true";  // 数据库 cau 的 JDBC URL
11             Connection con = DriverManager.getConnection(dbURL);  // 建立数据库连接
12             System.out.println(dbURL +" connected.");
13             // 创建学生表，包含学号、姓名、学院和班级，共 4 个字段
14             Statement s = con.createStatement();          // 首先创建语句对象
15             String sqlCreateTable = "CREATE TABLE student( " +  // SQL：创建新数据表
16                     "sNo CHAR(10) PRIMARY KEY, sName CHAR(10)," +
17                     "college VARCHAR(20), class VARCHAR(20) )";
18             s.executeUpdate(sqlCreateTable);  // 向 Java DB 提交 SQL 修改语句
19             System.out.println("TABLE student created.");
20             // 下面开始向学生表 student 插入 4 条新记录，分别保存 4 名同学的信息
21             // 插入保存第 1 名同学信息的记录
22             String sqlInsert = "INSERT INTO student VALUES( " +  // SQL：插入新记录
23                     "'20180001', '张同学', '信电学院', '计算 181' )";
24             s.executeUpdate(sqlInsert);  // 向 Java DB 提交 SQL 修改语句
25             // 插入保存第 2 名同学信息的记录
26             sqlInsert = "INSERT INTO student VALUES( " +
27                     "'20180002', '李同学', '信电学院', '计算 181' )";
28             s.executeUpdate(sqlInsert);
29             // 插入保存第 3 名同学信息的记录
30             sqlInsert = "INSERT INTO student VALUES( " +
31                     "'20180003', '王同学', '信电学院', '计算 182' )";
32             s.executeUpdate(sqlInsert);
33             // 插入保存第 4 名同学信息的记录
34             sqlInsert = "INSERT INTO student VALUES( " +
35                     "'20180004', '赵同学', '信电学院', '计算 182' )";
36             s.executeUpdate(sqlInsert);
37             System.out.println("4 student records inserted.");
38             // 数据库访问结束，关闭 SQL 语句对象和数据库连接对象
39             s.close();  con.close();
```

```
            }
            catch(ClassNotFoundException e) {    e.printStackTrace();    }   // 驱动程序加载异常
            catch(SQLException e) {    e.printStackTrace();    }                // SQL 语句执行异常
}   }
```

例 10-2 查询并显示学生表 student 中学生记录的 Java 示例程序（JDBSelect.java）

```
import java.sql.*;   // 导入 java.sql 包中数据库相关的类和接口
public class JDBSelect {                          // 主类：查询并显示学生表 student 中的学生记录
    public static void main(String[] args) {   // 主方法
        try {   // 处理数据库访问过程中可能出现的勾选异常
            String dbDriver = "org.apache.derby.jdbc.EmbeddedDriver";   // Java DB 驱动
            Class.forName(dbDriver);                                                  // 加载驱动
            System.out.println(dbDriver +" loaded.");
            String dbURL = "jdbc:derby:D:\\cau; create=true";   // 数据库 cau 的 JDBC URL
            Connection con = DriverManager.getConnection(dbURL);          // 连接数据库
            System.out.println(dbURL +" connected.");
            // 查询并显示出学生表 student 中的全部学生记录
            Statement s = con.createStatement();          // 创建语句对象
            String sqlSelect = "SELECT * FROM student";   // SQL 语句：查询全部记录
            ResultSet rs = s.executeQuery(sqlSelect);     // 向 Java DB 提交 SQL 查询语句
            // 查询语句会返回查询到的结果集 ResultSet，遍历并显示其中的记录
            System.out.println("学生表 student 中的全部记录如下：");
            while ( rs.next() ) {   // 移到下一记录，如到末尾则返回 false，循环结束
                System.out.print( rs.getString("sNo") +"\t" );      // 字段：学号 sNO
                System.out.print( rs.getString("sName") +"\t" );   // 字段：姓名 sName
                System.out.print( rs.getString("college") +"\t" );   // 字段：学院 college
                System.out.println( rs.getString("class") );         // 字段：班级 class
            }
            // 查询并显示出学生表 student 中“计算 181”班同学的学号和姓名
            sqlSelect = "SELECT sNo, sName FROM student " +   // SQL：查询学号和姓名
                        "WHERE class = '计算 181'";   // 查询条件：班级='计算 181'
            rs = s.executeQuery(sqlSelect);   // 向 Java DB 提交 SQL 查询语句
            System.out.println("学生表 student 中“计算 181”班同学的学号和姓名：");
            while ( rs.next() ) {   // 遍历并显示结果集 rs
                System.out.print( rs.getString("sNo") +"\t" );   // 字段：学号 sNO
```

```
30                          System.out.println( rs.getString("sName") );   // 字段：姓名 sName
31                  }
32                  // 数据库访问结束，关闭 SQL 语句对象和数据库连接对象
33                  s.close();   con.close();
34              }
35              catch(ClassNotFoundException e) {   e.printStackTrace();   }  // 驱动程序加载异常
36              catch(SQLException e) {   e.printStackTrace();   }              // SQL 语句执行异常
37  }  }
```

例 10-4 一个完整的测试用例类 JUnitATester 的示例代码（JDBUpdate.java）

```
1   import static org.junit.jupiter.api.Assertions.*;   // 导入 JUnit 相关的类库
2   import org.junit.jupiter.api.Test;
3
4   class JUnitATester {   // 测试用例类
5       /* 测试目的：检查待测试类 A 中各方法成员的算法是否正确
6        * 测试方法：使用断言 assertEquals()来检查实际输出与预期输出是否一致
7        */
8       @Test   // 注解：指定其后面的方法 testA()是一个在测试时需要被执行的方法
9       void testA() {              // 测试构造方法
10          A obj = new A(5);   // 新建一个对象 obj 并初始化为 5。初值 5 是测试用例的输入
11          assertEquals(obj.get(), 5);   // 调用 get()方法，使用断言检查实际输出是否为 5
12      }
13      @Test
14      void testSetGet() {                 // 测试设置数据方法 set()和读取数据方法 get()
15          A obj = new A(0);               // 新建一个对象 obj
16          obj.set(6);                     // 调用 set()方法，然后再调用 get()方法，检查结果
17          assertEquals(obj.get(), 6);   // 预期输出为 6，使用断言检查实际输出是否一致
18      }
19      @Test
20      void testGetSquare() {                      // 测试计算平方值方法 getSquare()
21          A obj = new A(7);                       // 新建一个对象 obj，并给一个初值 7
22          assertEquals(obj.getSquare(), 49);   // 预期输出 49，使用断言检查实际输出是否一致
23  }  }
```

补充第 11 章 网络爬虫

例 11-1 使用 GET 方法访问 URL 网页的示例代码（HTTPURLConnectionTest.java）

```
1  import java.net.*;   // 导入 java.net 网络包中的类
2  import java.io.*;    // 导入 java.io 输入输出流包中的类
3  import java.util.*;  // 导入映射接口 Map、HashMap
4
5  public class HTTPURLConnectionTest {
6      public static void main(String[] args) {
7          String htmlPage;
8          htmlPage = doGet("http://www.cau.edu.cn/index.html");
9          //htmlPage = doGet("https://www.baidu.com/s?wd=java");
10         if (htmlPage.length() <= 1000)   System.out.println(htmlPage);
11         else   System.out.println(htmlPage.substring(0,   1000));
12     }
13     public static String doGet(String httpURL) {
14         HttpURLConnection connection = null;
15         InputStream is = null;
16         BufferedReader br = null;
17         String htmlPage = null;   // 返回结果字符串
18         try {
19             URL url = new URL(httpURL);                          // 创建 URL 对象
20             connection = (HttpURLConnection)url.openConnection();  // 打开 URL 连接
21             // 先设置 HTTP 请求的头部信息，然后与 URL 建立连接
22             connection.setRequestMethod("GET");    // 设置连接方式：GET
23             connection.setConnectTimeout(15000);  // 设置连接 URL 的超时时间：15000 毫
24 秒
25             connection.setRequestProperty("User-Agent", "Mozilla/5.0");  // 设置浏览器名称
26             connection.connect();                                  // 与 URL 建立连接
27             // 以下代码用于处理所返回的 HTTP 响应
28             if (connection.getResponseCode() == 200) {   // 检查响应状态码
29               // 读取并显示 HTTP 响应的头部信息
30               System.out.println("------ HTTP response header -------");
31               Map<String,List<String>> hf = connection.getHeaderFields();
32               for   (Map.Entry<String,List<String>> entry : hf.entrySet() ) {
33                     String key = entry.getKey();
34                     List<String> vList = entry.getValue();
35                     for (String value : vList)
36                           System.out.println(key +" : " +value);
37               }
37               System.out.println("Content-Encoding : " +connection.getContentEncoding());
39               // 读取并显示 HTTP 响应正文部分的内容
```

```
                    System.out.println("------ HTTP response content ------");
                        is = connection.getInputStream();   // 获取 URL 连接的输入流
                        br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
                        StringBuffer sBuf = new StringBuffer();   // 定义存放网页的缓冲区
                        String sLine = null;
                        while ((sLine = br.readLine()) != null) {
                            sBuf.append(sLine);       sBuf.append("\r\n");
                        }
                        htmlPage = sBuf.toString();   // 将 StringBuffer 缓冲区传出字符串 String
                    }
                    br.close();   is.close();
                    connection.disconnect();   // 断开 URL 连接
                }
                catch (IOException e) {   e.printStackTrace();   }
                return htmlPage;   // 返回 URL 所指定的网页内容
    }   }
```

例 11-2 使用 POST 方法访问 URL 网页的示例代码（HTTPURLConnectionTest.java）

```
import java.net.*;   // 导入 java.net 网络包中的类
import java.io.*;    // 导入 java.io 输入输出流包中的类
import java.util.*;  // 导入映射接口 Map、HashMap

public class HTTPURLConnectionTest {
    public static void main(String[] args) {
        HashMap<String, String> params = new HashMap<String, String>();
        params.put("id", "1234");       params.put("name", "张同学");
        String htmlPage = doPost("http://www.cau.edu.cn/index.html", params);
        if (htmlPage.length() <= 1000)   System.out.println(htmlPage);
        else   System.out.println(htmlPage.substring(0,   1000));
    }
    public static String doPost(String httpURL, HashMap<String, String> params) {
        HttpURLConnection connection = null;
        OutputStream os = null;
        PrintWriter pw = null;
        InputStream is = null;
        BufferedReader br = null;
        String htmlPage = null;
        try {
            URL url = new URL(httpURL);                                    // 创建 URL 对象
            connection = (HttpURLConnection)url.openConnection();   // 打开 URL 连接
            // 先设置 HTTP 请求的头部信息，然后与 URL 建立连接
            connection.setRequestMethod("POST");   // 设置连接方式：POST
            connection.setConnectTimeout(15000);   // 设置连接 URL 的超时时间：15000 毫
秒
```

```
27                  connection.setRequestProperty("User-Agent", "Mozilla/5.0");   // 设置浏览器名称
28                  connection.setRequestProperty("Content-Type",
29                                                  "application/x-www-form-urlencoded");
30                  connection.setDoOutput(true);   // 默认值为 false，向 URL 上传数据时需设为 true
31                  // 上传参数：参数格式应该是 key1=value1&key2=value2 的形式
32                  os = connection.getOutputStream();   // 通过连接对象获取一个输出流
33                  pw = new PrintWriter(os);                  // 包装成字符流
34                  boolean addPrefix = false;
35                  for (Map.Entry<String,String> entry : params.entrySet() ) {
36                     if (addPrefix == false)    addPrefix = true;
37                     else    pw.print("&");
37                     String key = entry.getKey();       String value = entry.getValue();
39                     pw.print(key +"=" +URLEncoder.encode(value, "utf-8"));
40                  }
41                  pw.close();    os.close();
42                  // 以下代码用于处理所返回的 HTTP 响应
43                  if (connection.getResponseCode() == 200) {   // 检查响应状态码
44                     // 读取并显示 HTTP 响应的头部信息
45                     System.out.println("------ HTTP response header -------");
46                     Map<String,List<String>> hf = connection.getHeaderFields();
47                     for    (Map.Entry<String,List<String>> entry : hf.entrySet() ) {
48                           String key = entry.getKey();
49                           List<String> vList = entry.getValue();
50                           for (String value : vList)
51                                 System.out.println(key +" : " +value);
52                     }
53                     System.out.println("Content-Encoding : " +connection.getContentEncoding());
54                     // 读取并显示 HTTP 响应正文部分的内容
55                     System.out.println("------ HTTP response content ------");
56                        is = connection.getInputStream();   // 获取 URL 连接的字节型输入流
57                        br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
58                        StringBuffer sBuf = new StringBuffer();   // 定义存放网页的缓冲区
59                        String sLine = null;
60                        while ((sLine = br.readLine()) != null) {
61                           sBuf.append(sLine);       sBuf.append("\r\n");
62                        }
63                        htmlPage = sBuf.toString();   // 将 StringBuffer 缓冲区传出字符串 String
64                  }
65                  br.close();    is.close();
66                  connection.disconnect();   // 断开 URL 连接
67            }
68            catch (IOException e) {    e.printStackTrace();   }
69            return htmlPage;   // 返回 URL 所指定的网页内容
      }   }
```

例 11-3 使用 jsoup 库解析 html 网页的示例代码（JSoupTest.java）

```
import org.jsoup.*;              // 导入 org.jsoup 包中的类 Jsoup
import org.jsoup.nodes.*;   // 导入 org.jsoup.nodes 子包中的类 Document、Element 等
import java.io.*;                // 导入 java.io 包中的类 File

public class JSoupTest {
        public static void main(String[] args) {
                try {
                        /*
                        String html = "<html><head><title>First parse</title></head>" +
                                                "<body><p>Parsed HTML into a doc.</p></body></html>";
                        Document doc = Jsoup.parse(html);   // 解析 html 字符串，生成一个文档对象
                        */
                        File f = new File("data/test.html");
                        Document doc = Jsoup.parse(f, "gbk");   // 解析 html 文件，生成文档对象
                        System.out.println("tagName(): " + doc.tagName());
                        System.out.println("text(): " + doc.text());   // 提取文档的复合文本
                        System.out.println("------------------");
                        // 解析网页中的标签（tag），jsoup 将标签称为元素（Element）
                        Element ep = doc.selectFirst("p");   // 通过标签名提取标签，例如<p>标签
                        System.out.println("tagName(): " + ep.tagName());        // 提取标签的名称
                        System.out.println("className(): " + ep.className());   // 提取标签的 class 属性
                        System.out.println("text(): " + ep.text());                    // 提取标签的复合文本
                        System.out.println("ownText(): " + ep.ownText());         // 提取标签自己的文本
                        System.out.println("------------------");
                        // 通过 id 属性提取标签，例如：提取 id="link2"的标签
                        Element ea = doc.getElementById("link2");       // 提取所有 id="link2"的标签
                        //Element ea = doc.selectFirst("a#link2");        // 提取第一个 id="link2"的<p>标
签
                        //Element ea = doc.selectFirst("a[id=link2]");   // 提取第一个 id="link2"的<p>标
签
                        System.out.println("tagName(): " + ea.tagName());        // 提取标签的名称
                        System.out.println("className(): " + ea.className());   // 提取标签的 class 属性
                        System.out.println("text(): " + ea.text());                    // 提取标签的复合文本
                        System.out.println("ownText(): " + ea.ownText());         // 提取标签自己的文本
                }
               catch (IOException e) {    e.printStackTrace();    }
}   }
```

例 11-4 使用 jsoup 库从 html 网页中提取股票信息的示例程序（StockInfo.java）

```
1  import org.jsoup.*;              // 导入 org.jsoup 包中的类 Jsoup
2  import org.jsoup.nodes.*;   // 导入 org.jsoup.nodes 子包中的类 Document、Element 等
3  import org.jsoup.select.*;   // 导入 org.jsoup.select 包中的类 Jsoup
4  import java.net.*;               // 导入 java.net 包中的类 URL
5  import java.util.*;              // 导入 java.util 包中的类 ArrayList
6
7  public class StockInfo {
8          public static void main(String[] args) {
9                  try {
10                         URL url = new URL("https://gupiao.baidu.com/stock/sh600000.html");
11                         Document doc = Jsoup.parse(url, 60000);   // 下载并解析 url 指定的网页
12                         //System.out.println("tagName(): " + doc.tagName());
13                         // 提取网页中的股票信息：信息名、信息值
14                         ArrayList<String> key = new ArrayList<>();       // 保存信息名
15                         ArrayList<String> value = new ArrayList<>();   // 保存信息值
16                         // 查找 class 属性为"stock-bets"的<div>标签
17                         Element divInfo = doc.selectFirst("div.stock-bets");
18                         // 查找 class 属性为"bets-name"的<a>标签
19                         Element eName = divInfo.selectFirst("a.bets-name");   // 解析股票名称
20                         String sName = eName.ownText();     // 提取股票名称文本
21                         sName = sName.replace("()", "");   // 去除多余字符
22                         key.add("股票名称");       value.add(sName);
23                         // 查找 class 属性为"_close"的<strong>标签
24                         Element ePrice = divInfo.selectFirst("strong._close");   // 解析股票价格
25                         String sPrice = ePrice.ownText();   // 提取股票价格文本
26                         key.add("股票价格");       value.add(sPrice);
27                         // 查找所有的<dt>标签和<dd>标签
28                         Elements eDT = divInfo.select("dt");   // 解析其他股票信息
29                         Elements eDD = divInfo.select("dd");
30                         for (int n = 0; n < eDT.size(); n++) {   // 提取所查找到<dt>、<dd>标签中的文本
31                                 String k = eDT.get(n).ownText();   // 信息名
32                                 String v = eDD.get(n).ownText();   // 信息值
33                                 key.add(k);       value.add(v);
34                         }
35                         for (int n = 0; n < key.size(); n++)   // 显示所提取到的股票信息
36                                 System.out.println(key.get(n) +": " +value.get(n));
37                 }
38                 catch (Exception e) {   e.printStackTrace();   }
39 }  }
```

例 11-5 使用 jsoup 库从 html 网页中提取股票代码的示例程序（StockInfo.java）

```
1  import org.jsoup.*;            // 导入 org.jsoup 包中的类 Jsoup
2  import org.jsoup.nodes.*;   // 导入 org.jsoup.nodes 子包中的类 Document、Element 等
3  import org.jsoup.select.*;   // 导入 org.jsoup.select 包中的类 Jsoup
4  import java.net.*;             // 导入 java.net 包中的类 URL
5  import java.util.regex.*;     // 导入 java.util.regex 包中与正则表达式相关的类
6
7  public class StockList {
8        public static void main(String[] args) {
9              try {
10                   URL url = new URL("http://quote.eastmoney.com/stock_list.html");
11                   Document doc = Jsoup.parse(url, 60000);   // 下载并解析 url 指定的网页
12                   //System.out.println("tagName(): " + doc.tagName());
13                   // 提取网页中所有的股票代码：查找 class 属性为"quotebody"的<div>标签
14                   Element divInfo = doc.selectFirst("div.quotebody");
15                   // 指定 href 属性的正则表达式，然后查找所有符合条件的标签
16                   String regex = "a[href~=http.+s[hz][603]\\d{5}\\.html]";
17                   Elements eNo = divInfo.select(regex);
18                   // 遍历所查找到的标签：先提取标签的 href 属性，再提取其中的股票代码
19                   Pattern p = Pattern.compile("s[hz][603]\\d{5}");   // 股票代码的正则表达式
20                   for (int n = 0; n < eNo.size(); n++) {
21                         String h = eNo.get(n).attr("href");   // 提取标签中的 href 属性文本
22                         Matcher m = p.matcher(h);          // 使用正则表达式提取其中的股票代码
23                         if (m.find() == true) {                  // 存在符合正则表达式的股票代码
24                               int p1 = m.start(), p2 = m.end();
25                               System.out.println( h.substring(p1, p2) );
26                   }   }
27             }
28           catch (Exception e) {   e.printStackTrace();   }
29 }   }
```

例 11-6 使用 POI 库读取电子表格文件（xls、xlsx）的示例程序（POIxlsxTest.java）

```
1  import org.apache.poi.hssf.usermodel.*;  // 导入 xls 相关的类：HSSF*
2  import org.apache.poi.xssf.usermodel.*;  // 导入 xlsx 相关的类：XSSF*
3  import org.apache.poi.ss.usermodel.CellType;  // 导入枚举类型 CellType（单元格数据类型）
4  import java.util.Iterator;                    // 导入迭代器接口 Iterator
5  import java.io.*;                             // 导入文件相关的类
6
7  public class POIxlsxTest {
8      public static void main(String[] args) {
9          readXLS("data/test1.xls");      // 读取 xls 电子表格文件
10         readXLSX("data/test2.xlsx");    // 读取 xlsx 电子表格文件
11     }
12
13     public static void readXLS(String filename) {   // 读取 xls 电子表格文件
14         try {
15             InputStream xlsFile = new FileInputStream(filename);   // xls 文件输入流对象
16             HSSFWorkbook wb = new HSSFWorkbook(xlsFile);   // 从 xls 文件创建工作簿对象
17             HSSFSheet sheet = wb.getSheetAt(0);                // 读取第一张工作表
18             HSSFRow row;
19             HSSFCell cell;
20             Iterator itRow = sheet.rowIterator();     // 取得工作簿的行迭代器
21             while ( itRow.hasNext() ) {               // 遍历工作表的每一行
22                 row = (HSSFRow) itRow.next();       // 读取下一行
23                 Iterator itCell = row.cellIterator();   // 取得行的单元格迭代器
24                 while ( itCell.hasNext() ) {            // 遍历一行中的所有单元格
25                     cell = (HSSFCell) itCell.next();   // 读取下一个单元格
26                     if (cell.getCellType() == CellType.STRING)          // 文本类型的单元格
27                         System.out.print(cell.getStringCellValue() + " ");
28                     else if (cell.getCellType() == CellType.NUMERIC)   // 数值类型的单元格
29                         System.out.print(cell.getNumericCellValue() + " ");
30                 }
31                 System.out.println();
32             }
33             wb.close();   // 关闭工作簿
34         }   catch (Exception e) {   e.printStackTrace();   }
35     }
36
37     public static void readXLSX(String filename) {   // 读取 xlsx 电子表格文件
38         try {
39             InputStream xlsxFile = new FileInputStream(filename);   // xlsx 文件输入流对象
40             XSSFWorkbook wb = new XSSFWorkbook(xlsxFile);   // 从 xlsx 文件创建工作簿对象
41             XSSFSheet sheet = wb.getSheetAt(0);                // 读取第一张工作表
42             XSSFRow row;
43             XSSFCell cell;
```

```
44                  Iterator itRow = sheet.rowIterator();     // 取得工作簿的行迭代器
45                  while ( itRow.hasNext() ) {                  // 遍历工作表的每一行
46                      row = (XSSFRow) itRow.next();       // 读取下一行
47                      Iterator itCell = row.cellIterator();   // 取得行的单元格迭代器
48                      while ( itCell.hasNext() ) {              // 遍历一行中的所有单元格
49                          cell = (XSSFCell) itCell.next();   // 读取下一个单元格
50                          if (cell.getCellType() == CellType.STRING)              // 文本类型的单元格
51                              System.out.print(cell.getStringCellValue() + " ");
52                          else if (cell.getCellType() == CellType.NUMERIC)    // 数值类型的单元格
53                              System.out.print(cell.getNumericCellValue() + " ");
54                      }
55                      System.out.println();
56                  }
57                  wb.close();   // 关闭工作簿
58              }   catch (Exception e) {    e.printStackTrace();   }
59  }   }
```