# 项目三

# 记事本应用程序开发

## 项目目标

完成一个记事本应用程序的开发，能够实现文字的编辑，各类字符分类统计，文件的打开和保存。通过本项目掌握 Menu 类、MenuItem 类、MenuBar 类和 TextArea 类的方法；掌握包装类和字符串 String 类的作用和常用方法；掌握输入输出流的读写文本文件的方法。

## 任务一　记事本界面设计

### 【任务描述】

完成记事本应用程序的界面设计，包括菜单设计。

### 【任务分析】

记事本的界面主要包含一个文本区域和多个菜单。

本任务的关键点：

菜单的创建和设置。

### 【预备知识】

在 GUI 应用程序中，菜单是非常有用的。菜单将显示一列菜单项，指明用户可以执行的各项操作。使用菜单的方式很简单，菜单通常会显示已粗略分类的数个选项。选择或单击某个菜单就会打开另一列菜单或菜单项。每个菜单项都将有一些与之关联的操作。菜单放在菜单栏中，菜单项放在菜单里。

1. Menu 类

Menu 类是一个菜单类，它的对象可以在菜单栏显示文本字符串，而当用户单击此字符串时，则显示一个弹出式菜单。其继承关系如图 3-1 所示。

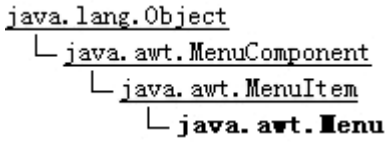


图 3-1　Menu 类的继承关系图

Menu 类的构造方法见表 3.1。

表 3.1　Menu 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| Menu() | 构造具有空标签的新菜单 |
| Menu(String label) | 构造具有指定标签的新菜单 |

Menu 类的常用方法见表 3.2。

表 3.2　Menu 类的常用方法

| 方法 | 说明 |
| --- | --- |
| MenuItem add(MenuItem mi) | 将指定的菜单项添加到此菜单 |
| void addSeparator() | 将一个分隔线或连字符添加到菜单的当前位置 |

2. MenuItem 类

MenuItem 类负责创建菜单项，即 MenuItem 类的对象是一个菜单项。其继承关系如图 3-2 所示。

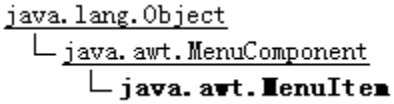


图 3-2　MenuItem 类的继承关系图

MenuItem 类的构造方法见表 3.3。

表 3.3　MenuItem 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| MenuItem() | 构造具有空标签且没有键盘快捷方式的新菜单项 |
| MenuItem(String label) | 构造具有指定的标签且没有键盘快捷方式的新菜单项 |
| MenuItem(String label, MenuShortcut s) | 创建具有关联的键盘快捷方式的菜单项 |

MenuItem 类的常用方法见表 3.4。

表 3.4　MenuItem 类的常用方法

| 方法 | 说明 |
| --- | --- |
| String getLabel() | 获取此菜单项的标签 |
| void setEnable(boolean b) | 设置是否可以选择此菜单项 |

3. MenuBar 类

MenuBar 类是负责创建菜单栏的，即 MenuBar 的一个对象就是一个菜单栏。其继承关系如图 3-3 所示。Frame 类有一个将菜单栏放置到窗口中的方法：

```
void setMenuBar(MenuBar bar)
```

该方法将菜单栏添加到窗口的顶端，需要注意的是，只能向窗口添加一个菜单栏。

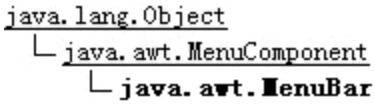


图 3-3　MenuBar 类的继承关系图

MenuBar 类的构造方法见表 3.5。

表 3.5　MenuBar 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| MenuBar() | 创建新的菜单栏 |

MenuBar 类的常用方法见表 3.6。

表 3.6　MenuBar 类的常用方法

| 方法 | 说明 |
| --- | --- |
| MenuBar add(Menu m) | 将指定的菜单添加到菜单栏 |

例 3.1　创建一个带菜单的窗口。

```
import java.awt.*;
class FirstWindow extends Frame
{
    //声明菜单栏，菜单，菜单项
    MenuBar menubar;
     Menu menu;
     MenuItem    item1,item2;
    //构造方法
    FirstWindow(String s)
     { //设置窗口标题
        setTitle(s);
        //设置边界
        setBounds(0,0,dim.width,dim.height/2);
        //实例化菜单栏
        menubar=new MenuBar();
        //实例化菜单
        menu=new Menu("文件");
        //实例化菜单项
        item1=new MenuItem("打开");
        item2=new MenuItem("保存");
        //将菜单项添加到菜单
        menu.add(item1);
        menu.add(item2);
        //将菜单添加菜单栏
        menubar.add(menu);
```

```
            //将菜单栏添加到窗口
            setMenuBar(menubar);
            setVisible(true);
        }
}
public class Example2_1
{    public static void main(String args[])
        {   FirstWindow win=new FirstWindow("一个带菜单的窗口");
        }
}
```

## 【任务实施】

完成记事本，可以参考 Windows 的记事本应用程序，如图 3-4 所示。


图 3-4　Windows 的记事本界面

完成这个界面设计，需要用到一个窗口 Frame，一个菜单栏 MenuBar，n 个菜单 Menu，n 个菜单项，一个文本区。

（1）编写 MainFrame 类，该类为记事本文件的窗口，在窗口中添加文本区 TextArea 对象，创建菜单及菜单项。

```
import java.awt.Frame;
import java.awt.Menu;
import java.awt.MenuBar;
import java.awt.MenuItem;
import java.awt.TextArea;

public class MainFrame extends Frame{
    TextArea ta;
    MenuItem miOpen,miNew,miSave,miSaveAs,miClose,miCopy,miPaste,miCut,miHelp;
    Menu mFile,mEdit,mHelp;
    MenuBar mb;
    String s;
    MainFrame(){
        ta=new TextArea();
        miOpen=new MenuItem("打开");
        miNew=new MenuItem("新建");
        miSave=new MenuItem("保存");
        miSaveAs=new MenuItem("另存为");
```

```
        miClose=new MenuItem("关闭");
        miCopy=new MenuItem("复制");
        miPaste=new MenuItem("粘贴");
        miCut=new MenuItem("剪切");
        miHelp=new MenuItem("帮助");


        mFile=new Menu("文件");
        mEdit=new Menu("编辑");
        mHelp=new Menu("帮助");

        mb=new MenuBar();

        mFile.add(miNew);
        mFile.add(miOpen);
        mFile.insertSeparator(2);
        mFile.add(miSave);
        mFile.add(miSaveAs);
        mFile.add(miClose);

        mEdit.add(miCopy);
        mEdit.add(miPaste);
        mEdit.add(miCut);

        mHelp.add(miHelp);

        mb.add(mFile);
        mb.add(mEdit);
        mb.add(mHelp);

        this.add(ta);
        this.setMenuBar(mb);

    }
}
```

（2）新建一个包含 main 方法的测试类，在该类中实例化记事本主窗口。

```
public class Test {
    public static void main(String[] args) {
        MainFrame f=new MainFrame();
        f.setTitle("记事本");
        f.setSize(400, 400);
        f.setVisible(true);
    }

}
```

## 【任务小结】

通过本任务，完成对记事本的界面设计，主要掌握菜单的设计。

## 【思考与习题】

1．下列容器是从 java.awt.Window 继承的是（　　）。

A．Applet　　B．Panel　　C．Container　　D．Frame

2．在 Java 图形用户界面编程中，若显示一些需要添加或修改的单行文本信息，一般是使用（　　）类的对象来实现。

A．Label　　B．Button　　C．TexTarea　　D．TestField

3．菜单组成的基本要素包括（　　）。

A．菜单栏　　B．菜单框　　C．菜单　　D．菜单项

4．请在已经完成的记事本程序上添加一个菜单：“格式”，包含两个菜单项“字体”和“颜色”。

## 任务二　记事本的文本编辑功能

### 【任务描述】

完成记事本的文本编辑功能，包括复制、粘贴、剪切、分类统计字母、数字和字符的个数。

### 【任务分析】

本任务的关键点：

- 字符串类的使用。
- 包装类的使用。
- 文本区域组件的常用方法。

### 【预备知识】

1．字符串类

在 Java 中，字符串常量不同于基本数据类型，而是被看做一个 String 类的对象。因此，可以通过使用 String 类提供的方法完成对字符串的操作。创建一个字符串对象之后，将不能更改构成字符串的字符。其继承关系如图 3-5 所示。

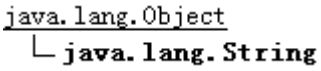


图 3-5　String 的继承关系图

（1）String 对象的初始化

由于 String 对象非常常用，所以在对 String 对象进行初始化时，Java 提供了一种简化的特殊语法，格式如下：

```
String s = "abc";
s = "Java 语言";
```

其实按照面向对象的标准语法，其格式应该为：

```
String s = new String("abc");
       s = new String("Java 语言");
```

只是按照面向对象的标准语法，在内存使用上存在比较大的浪费。例如 String s = new String("abc"); 实际上创建了两个 String 对象，一个是"abc"对象，存储在常量空间中，一个是使用 new 关键字为对象 s 申请的空间。

String 类的构造方法见表 3.7。

表 3.7　String 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| String() | 初始化一个新创建的 String 对象，使其表示一个空字符序列 |
| String(byte[] bytes) | 通过使用平台默认字符集解码指定的 byte 数组，构造一个新的 String |
| String(byte[] bytes, int offset, int length) | 通过使用平台默认字符集解码指定的 byte 子数组，构造一个新的 String |
| String(char[] value) | 分配一个新的 String，使其表示字符数组参数中当前包含的字符序列 |
| String(char[] value, int offset, int count) | 分配一个新的 String，它包含取自字符数组参数一个子数组的字符 |
| String(String original) | 初始化一个新创建的 String 对象，使其表示一个与参数相同的字符序列；换句话说，新创建的字符串是该参数字符串的副本 |
| String(StringBuffer buffer) | 分配一个新的字符串，它包含字符串缓冲区参数中当前包含的字符序列 |

（2）字符串的常见操作

1）charAt 方法

该方法的作用是按照索引值（规定字符串中第一个字符的索引值是 0，第二个字符的索引值是 1，依次类推），获得字符串中的指定字符。例如：

```
String s = "abc";
char c = s.chatAt(1);
```

则变量 c 的值是'b'。

2）compareTo 方法

该方法的作用是比较两个字符串的大小，比较的原理是依次比较每个字符的字符编码。首先比较两个字符串的第一个字符，如果第一个字符串的字符编码大于第二个字符串的字符编码，则返回大于 0 的值，如果小于则返回小于 0 的值，如果相等则比较后续的字符，如果两个字符串中的字符编码完全相同则返回 0。

例如：

```
String s = "abc";
String s1 = "abd";
int value = s.compareTo(s1);
```

则 value 的值是小于 0 的值，即–1。

在 String 类中还存在一个类似的方法 compareToIgnoreCase，这个方法是忽略字符的大小写进行比较，比较的规则和 compareTo 一样。例如：

```
String s = "aBc";
String s1 = "ABC";
int value = s. compareToIgnoreCase (s1);
```

则 value 的值是 0，即两个字符串相等。

3）concat 方法

该方法的作用是进行字符串的连接，将两个字符串连接以后形成一个新的字符串。例如：

```
String s = "abc";
String s1 = "def";
String s2 = s.concat(s1);
```

则连接以后生成的新字符串 s2 的值是“abcdef”，而字符串 s 和 s1 的值不发生改变。如果需要连接多个字符串，可以使用如下方法：

```
String s = "abc";
String s1 = "def";
String s2 = "1234";
String s3 = s.concat(s1).concat(s2);
```

则生成的新字符串 s3 的值为“abcdef1234”。

其实在实际使用时，语法上有一种更简单的形式，就是使用“+”进行字符串的连接。例如：

```
String s = "abc" + "1234";
```

则字符串 s 的值是“abc1234”，这样书写更加简单直观。而且使用“+”进行连接，不仅可以连接字符串，也可以连接其他类型。但是要求进行连接时至少有一个参与连接的内容是字符串类型。而且“+”匹配的顺序是从左向右，如果两边连接的内容都是基本数字类型则按照加法运算，如果参与连接的内容至少有一个是字符串才按照字符串进行连接。

例如：

```
int a = 10;
String s = "123" + a + 5;
```

则连接以后字符串 s 的值是“123105”，计算的过程为首先连接字符串“123”和变量 a 的值，生成字符串“12310”，然后使用该字符串再和数字 5 进行连接生成最终的结果。

而如下代码：

```
int a = 10;
String s = a + 5 + "123";
```

则连接以后字符串 s 的值是“15123”，计算的过程为首先计算 a 和数字 5，由于都是数字型则进行加法运算得到数字 15，然后再使用数字 15 和字符串“123”进行连接获得最终的结果。

而下面的连接代码是错误的：

```
int a = 12;
String s = a + 5 + 's';
```

因为参与连接的没有一个是字符串，则计算出来的结果是数字值，在赋值时无法将一个数字值赋值给字符串 s。

4）endsWith 方法

该方法的作用是判断字符串是否以某个字符串结尾，如果以对应的字符串结尾，则返回 true。

例如：

```
String s = "student.doc";
boolean b = s.endsWith("doc");
```

则变量 b 的值是 true。

5）equals 方法

该方法的作用是判断两个字符串对象的内容是否相同。如果相同则返回 true，否则返回 false。

例如：

```
String s = "abc";
String s1 = new String("abc");
boolean b = s.equals(s1);
```

而使用“==”比较的是两个对象在内存中存储的地址是否一样。例如上面的代码中，如果判断：

```
boolean b = (s == s1);
```

则变量 b 的值是 false，因为 s 对象对应的地址是“abc”的地址，而 s1 使用 new 关键字申请新的内存，所以内存地址和 s 的“abc”的地址不一样，所以获得的值是 false。

在 String 类中存在一个类似的方法 equalsIgnoreCase，该方法的作用是忽略大小写比较两个字符串的内容是否相同。例如：

```
String s = "abc";
String s1 ="ABC";
boolean b = s. equalsIgnoreCase (s1);
```

则变量 b 的值是 true。

6）getBytes 方法

该方法的作用是将字符串转换为对应的 byte 数组，从而便于数据的存储和传输。例如：

```
String s = "计算机";
byte[] b = s.getBytes();    //使用本机默认的字符串转换为 byte 数组
byte[] b = s.getBytes("gb2312");    //使用 gb2312 字符集转换为 byte 数组
```

在实际转换时，一定要注意字符集的问题，否则中文在转换时将会出现问题。

7）indexOf 方法

该方法的作用是查找特定字符或字符串在当前字符串中的起始位置，如果不存在则返回–1。例如：

```
String s = "abcded";
int index = s.indexOf('d');
int index1 = s.indexOf('h');
```

则返回字符 d 在字符串 s 中第一次出现的位置，数值为 3。由于字符 h 在字符串 s 中不存在，则 index1 的值是–1。

当然，也可以从特定位置以后查找对应的字符，例如：

```
int index = s.indexOf('d',4);
```

则查找字符串 s 中从索引值 4（包括 4）以后的字符中第一个出现的字符 d，则 index 的值是 5。

由于 indexOf 是重载的，也可以查找特定字符串在当前字符串中出现的起始位置，使用方式和查找字符的方式一样。

另外一个类似的方法是 lastIndexOf 方法，其作用是从字符串的末尾开始向前查找第一次出现的规定字符或字符串，例如：

```
String s = "abcded";
int index = s. lastIndexOf('d');
```

则 index 的值是 5。

8）length 方法

该方法的作用是返回字符串的长度，也就是返回字符串中字符的个数。中文字符也是一个字符。例如：

```
String s = "abc";
String s1 = "Java 语言";
int len = s.length();
int len1 = s1.length();
```

则变量 len 的值是 3，变量 len1 的值是 6。

9）replace 方法

该方法的作用是替换字符串中所有指定的字符，然后生成一个新的字符串。经过该方法调用以

后，原来的字符串不发生改变。例如：

```
String s = "abcat";
String s1 = s.replace('a','1');
```

该代码的作用是将字符串 s 中所有的字符 a 替换成字符 1，生成的新字符串 s1 的值是“1bc1t”，而字符串 s 的内容不发生改变。

如果需要将字符串中某个指定的字符串替换为其他字符串，则可以使用 replaceAll 方法，例如：

```
String s = "abatbac";
String s1 = s.replaceAll("ba","12");
```

该代码的作用是将字符串 s 中所有的字符串“ab”替换为“12”，生成新的字符串“a12t12c”，而字符串 s 的内容也不发生改变。

如果只需要替换第一个出现的指定字符串时，可以使用 replaceFirst 方法，例如：

```
String s = "abatbac";
String s1 = s. replaceFirst ("ba","12");
```

该代码的作用是只将字符串 s 中第一次出现的字符串“ab”替换为字符串“12”，则字符串 s1 的值是"a12tbac"，而字符串 s 的内容不发生改变。

10）split 方法

该方法的作用是以特定的字符串作为间隔，拆分当前字符串的内容，一般拆分以后会获得一个字符串数组。例如：

```
String s = "ab,12,df";
String s1[] = s.split(',');
```

该代码的作用是以字符“,”作为间隔，拆分字符串 s，从而得到拆分以后的字符串数组 s1，其内容为：{"ab","12","df"}。

该方法是解析字符串的基础方法。

如果在字符串内部存在和间隔字符串相同的内容时将拆除空字符串，尾部的空字符串会被忽略。例如：

```
String s = "abbcbtbb";
String s1[] = s.split('b');
```

则拆分出的结果字符串数组 s1 的内容为：{'a','','c','t'}。拆分出的中间的空字符串的数量等于中间间隔字符串的数量减一。例如：

```
String s = "abbbcbtbbb";
String s1[] = s.split('b');
```

则拆分出的结果是：{'a','','','c','t'}。最后的空字符串不论有多少个，均被忽略。

如果需要限定拆分以后的字符串数量，则可以使用另外一个 split 方法，例如：

```
String s = "abcbtb1";
String s1[] = s.split('b',2);
```

该代码的作用是将字符串 s 最多拆分成 2 个字符串数组。则结果为：{"a","cbtb1"}。

如果第二个参数为负数，则拆分出尽可能多的字符串，包括尾部的空字符串也将被保留。

11）startsWith 方法

该方法的作用和 endsWith 方法类似，只是该方法是判断字符串是否以某个字符串作为开始。例如：

```
String s = "TestGame";
boolean b = s.startsWith("Test");
```

则变量 b 的值是 true。

12）substring 方法

该方法的作用是取字符串中的“子串”，所谓“子串”即字符串中的一部分。例如“23”是字符串“123”的子串。

字符串"123"的子串一共有 6 个："1"、"2"、"3"、"12"、"23"、"123"。而"32"不是字符串"123"的子串。

例如：

```
String s = "Test";
String s1 = s.substring(2);
```

则该代码的作用是取字符串 s 中索引值为 2（包括）以后的所有字符作为子串，则字符串 s1 的值是"st"。

如果数字的值和字符串的长度相同，则返回空字符串。例如：

```
String s = "Test";
String s1 = s.substring(4);
```

则字符串 s1 的值是""。

如果需要取字符串内部的一部分，则可以使用带 2 个参数的 substring 方法，例如：

```
String s = "TestString";
String s1 = s.substring(2,5);
```

则该代码的作用是取字符串 s 中从索引值 2（包括）开始，到索引值 5（不包括）的部分作为子串，则字符串 s1 的值是"stS"。

下面是一个简单的应用代码，该代码的作用是输出任意一个字符串的所有子串。代码如下：

```
String s = "子串示例";
int len = s.length(); //获得字符串长度
for(int begin = 0;begin < len – 1;begin++){ //起始索引值
        for(int end = begin + 1;end <= len;end++){ //结束索引值
                System.out.println(s.substring(begin,end));
        }
}
```

在该代码中，循环变量 begin 代表需要获得的子串的起始索引值，其变化的区间从第一个字符的索引值 0 到倒数第二个字符的索引值 len –2，而 end 代表需要获得的子串的结束索引值，其变化的区间从起始索引值的后续一个到字符串长度。通过循环嵌套，可以遍历字符串中的所有子串。

13）toCharArray 方法

该方法的作用和 getBytes 方法类似，即将字符串转换为对应的 char 数组。例如：

```
String s = "abc";
char[] c = s.toCharArray();
```

则字符数组 c 的值为：{'a','b','c'}。

14）toLowerCase 方法

该方法的作用是将字符串中所有大写字符都转换为小写字符。例如：

```
String s = "AbC123";
String s1 = s.toLowerCase();
```

则字符串 s1 的值是"abc123"，而字符串 s 的值不变。

类似的方法是 toUpperCase，该方法的作用是将字符串中的小写字符转换为对应的大写字符。

例如：

```
String s = "AbC123";
String s1 = s. toUpperCase ();
```

则字符串 s1 的值是"ABC123"，而字符串 s 的值也不变。

15）trim 方法

该方法的作用是去掉字符串开始和结尾的所有空格，然后形成一个新的字符串。该方法不去掉字符串中间的空格。例如：

```
String s = "   abc abc 123 ";
String s1 = s.trim();
```

则字符串 s1 的值为："abc abc 123"。字符串 s 的值不变。

16）valueOf 方法

该方法的作用是将其他类型的数据转换为字符串类型。需要注意的是：基本数据和字符串对象之间不能使用以前的强制类型转换的语法进行转换。

另外，由于该方法是 static 方法，所以不用创建 String 类型的对象。例如：

```
int n = 10;
String s = String.valueOf(n);
```

则字符串 s 的值是"10"。虽然对于程序员来说，没有发生什么变化，但是对于程序来说，数据的类型却发生了变化。

介绍一个简单的应用，判断一个自然数是几位数字的逻辑代码如下：

```
int n = 12345;
String s = String.valueOf(n);
int len = s.length();
```

这里字符串的长度 len，就代表该自然数的位数。这种判断比数学判断方法在逻辑上要简单一些。

**例 3.2** 中国古代有一种对联方式称为“回文”，如“人上天然居，居然天上人”，“人过大佛寺，寺佛大过人”。编写程序完成字符串的倒序输出。

```
public class StringInvert{
    public static void main(String []args){
        String s="人上天然居";
        String result;
        for(int i=(s.length-1);i>=0;i--){
            result=result.append(s.chartAt (i));
        }
        System.out.println(result);
    }
}
```

2. 包装类

Java 语言是一个面向对象的语言，但是 Java 中的基本数据类型却不是面向对象的，这在实际使用时存在很多不便，为了解决这个不足，在设计类时为每个基本数据类型设计了一个对应的类进行代表，这样 8 个和基本数据类型对应的类统称为包装类（Wrapper Class），有些地方也翻译为外覆类或数据类型类。

包装类均位于 java.lang 包中，包装类和基本数据类型的对应关系如表 3.8 所示。

在这 8 个类名中，除了 Integer 和 Character 类，其他 6 个类的类名和基本数据类型一致，只是类名的第一个字母大写即可。

对于包装类说，这些类的用途主要包含两种：

①作为和基本数据类型对应的类类型存在，方便涉及到对象的操作。

②包含每种基本数据类型的相关属性，如最大值、最小值等，以及相关的操作方法。

表 3.8　基本数据类型与包装类的对映表

| 基本数据类型 | 包装类 |
| --- | --- |
| byte | Byte |
| boolean | Boolean |
| short | Short |
| char | Character |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

由于 8 个包装类的使用比较类似，下面以最常用的 Integer 类为例介绍包装类的实际使用。

（1）实现基本数据类型和包装类之间的转换

在实际转换时，使用 Integer 类的构造方法和 Integer 类内部的 intValue 方法实现这些类型之间的相互转换，实现的代码如下：

```
int n = 10;
Integer in = new Integer(100);
//将 int 类型转换为 Integer 类型
Integer in1 = new Integer(n);
//将 Integer 类型的对象转换为 int 类型
int m = in.intValue();
```

（2）包装类内部的常用方法

在 Integer 类内部包含了一些和 int 操作有关的方法，下面介绍一些比较常用的方法：

1）parseInt 方法

```
public static int parseInt(String s)
```

该方法的作用是将数字字符串转换为 int 数值。在后续的界面编程中，将字符串转换为对应的 int 数字是一种比较常见的操作。使用示例如下：

```
String s = "123";
int n = Integer.parseInt(s);
```

则 int 变量 n 的值是 123，该方法实际上实现了字符串和 int 之间的转换，如果字符串包含的不都是数字字符，程序执行将会出现异常。

另外一个 parseInt 方法：

```
public static int parseInt(String s, int radix)
```

则实现将字符串按照参数 radix 指定的进制转换为 int，使用示例如下：

```
//将字符串"120"按照十进制转换为 int，则结果为 120
int n = Integer.parseInt("120",10);
//将字符串"12"按照十六进制转换为 int，则结果为 18
int n = Integer.parseInt("12",16);
//将字符串"ff"按照十六进制转换为 int，则结果为 255
int n = Integer.parseInt("ff",16);
```

这样可以实现更灵活的转换。

2）toString 方法

```
public static String toString(int i)
```

该方法的作用是将 int 类型转换为对应的 String 类型。

使用示例代码如下：

```
int m = 1000;
String s = Integer.toString(m);
```

则字符串 s 的值是"1000"。

另外一个 toString 方法则实现将 int 值转换为特定进制的字符串：

```
public static int parseInt(String s, int radix)
```

使用示例代码如下：

```
int m = 20;
String s = Integer.toString(m);
```

则字符串 s 的值是"14"。

其实，JDK 从 1.5(5.0)版本以后，就引入了自动拆装箱的语法，也就是在进行基本数据类型和对应的包装类转换时，系统将自动进行，这将大大方便程序员的代码书写。使用示例代码如下：

```
//int 类型会自动转换为 Integer 类型
int m = 12;
Integer in = m;
//Integer 类型会自动转换为 int 类型
int n = in;
```

所以在实际使用时类型转换将变得很简单，系统将自动实现对应的转换。

（3）包装类 Character

Character 类在对象中包装一个基本类型 char 的值。其继承关系如图 3-6 所示。Character 类型的对象包含类型为 char 的单个字段。

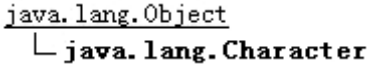


图 3-6 Character 类的继承关系图

Character 类的构造方法见表 3.9。

表 3.9 Character 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| Character(char value) | 构造一个新分配的 Character 对象，用以表示指定的 char 值 |

Character 类的常用方法见表 3.10。

表 3.10 Character 类的常用方法

| 常用方法 | 说明 |
| --- | --- |
| boolean equals(Object obj) | 将此对象与指定对象比较。当且仅当参数不为 null，而是一个与此对象包含相同 char 值的 Character 对象时，结果才是 true |
| static boolean isDigit(char ch) | 确定指定字符是否为数字 |
| static boolean isLetter(char ch) | 确定指定字符是否为字母 |
| static boolean isLowerCase(char ch) | 确定指定字符是否为小写字母 |
| static boolean isSpaceChar(char ch) | 确定指定字符是否为 Unicode 空白字符 |
| static boolean isTitleCase(char ch) | 确定指定字符是否首字母为大写字符 |

续表

| 常用方法 | 说明 |
| --- | --- |
| static boolean isUpperCase(char ch) | 确定指定字符是否为大写字母 |
| static char toUpperCase(char ch) | 使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为大写 |

例 3.3　判断一个字符串中有多少个字符，多少个字母，多少个数字。

```
public class Example2_3{
    public static void main(String []args){
        String s="123 I have a dream 456";
        System.out.println(s+"中一共有"+s.length()+"个字符");
        int numDigit=0,numLetter=0;
        for(int i=0;i<s.length();i++){
            char a=s.charAt(i);
            if(Character.isDigit(a)){
              numDigit=numDigit+1;
            }
            if(Character.isLetter(a)){
              numLetter=numLetter+1;
            }
        }
    }
}
```

3. 文本区域 TextArea

TextArea 对象是显示文本的多行区域。可以将它设置为允许编辑或只读。其继承关系如图 3-7 所示。

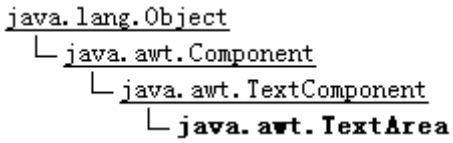


图 3-7　TextArea 类的继承关系图

TextArea 类的构造方法见表 3.11。

表 3.11　TextArea 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| TextArea() | 构造一个将空字符串作为文本的新文本区 |
| TextArea(int rows, int columns) | 构造一个新文本区，该文本区具有指定的行数和列数，并将空字符串作为文本 |
| TextArea(String text) | 构造具有指定文本的新文本区 |
| TextArea(String text, int rows, int columns) | 构造一个新文本区，该文本区具有指定的文本，以及指定的行数和列数 |
| TextArea(String text, int rows, int columns, int scrollbars) | 构造一个新文本区，该文本区具有指定的文本，以及指定的行数、列数和滚动条可见性 |

TextArea 类的常用方法见表 3.12。

表 3.12 TextArea 类的常用方法

| 方法 | 说明 |
| --- | --- |
| void append(String str) | 将给定文本追加到文本区的当前文本 |
| void insert(String str,int pos) | 在此文本区的指定位置插入指定文本 |
| void replaceRange(String str, int start,int end) | 用指定文本替换指定开始位置与结束位置之间的文本。结束处的文本不会被替换 |
| String getSelectedText() | 返回此文本组件所表示文本的选定文本 |
| int getSelectionEnd() | 获取此文本组件中选定文本的结束位置 |
| void setSelectionEnd( int selectionEnd) | 将此文本组件的选定结束位置设置为指定位置。新的结束点限制在当前选定开始位置处或之后。并且不能将它设置为超出组件文本末端 |
| void select(int selectionStart, int selectionEnd) | 选择指定开始位置和结束位置之间的文本 |
| String getText() | 返回此文本组件表示的文本。默认情况下，此文本是一个空字符串 |
| void setText(String t) | 将此文本组件显示的文本设置为指定文本 |

【任务实施】

```
import java.awt.Frame;
import java.awt.Menu;
import java.awt.MenuBar;
import java.awt.MenuItem;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

public class MainFrame extends Frame implements ActionListener{
    TextArea ta;
    MenuItem miOpen,miNew,miSave,miSaveAs,miClose,miCopy,miPaste,miCut,miHelp;
    Menu mFile,mEdit,mHelp;
    MenuBar mb;
    String s;
    MainFrame(){
        ta=new TextArea();
        …
        this.add(ta);
        this.setMenuBar(mb);

        this.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        };
        miClose.addActionListener(this);
        miCopy.addActionListener(this);
```

```
            miPaste.addActionListener(this);
            miCut.addActionListener(this);
            s=null;
        }
            public void actionPerformed(ActionEvent e) {
            if(e.getSource()==miClose){
                System.exit(0);
            }
            if(e.getSource()==miNew){
                ta.setText("");
            }
            if(e.getSource()==miCopy){
                s=ta.getSelectedText();
            }
            if(e.getSource()==miPaste){
                int pos=ta.getCaretPosition();
                ta.insert(s, pos);
            }
            if(e.getSource()==miCut){
                s=ta.getSelectedText();
                int st=ta.getSelectionStart();
                int en=ta.getSelectionEnd();
                ta.replaceRange("", st, en);
            }
        }
}
```

【任务小结】

通过本任务，学习 String 类和包装类的使用及文本区域组件的常用方法。实现记事本程序的文本编辑功能。

【思考与习题】

1．以下 Character 类的方法中，（　　）可以确定字符是否为字母。

A．isDigit()方法　　B．isLetter()方法

C．isSpace()方法　　D．isUnicodeIdentifier()方法

2．Java 提供名为（　　）的包装类来包装基本数据类型 int。

A．Integer　　B．Double　　C．String　　D．Char

3．下列 String 类的（　　）方法返回指定字符串的一部分。

A．extractstring()　　B．substring()　　C．Substring()　　D．middlestring()

4．java.lang 包的（　　）方法将比较两个对象是否相等，如果相等则返回 true。

A．toString()　　B．compare()　　C．equals()　　D．以上选项都不正确

## 任务三　完成对话框

【任务描述】

完成菜单项的“帮助”对话框。

【任务分析】

单击“帮助”菜单后，应弹出一个对话框，该对话框的内容由程序开发人员自定，可以提示一定信息，也可以返回一些数据。

本任务的关键点：

- 对话框的声明和使用。
- 文件对话框、颜色对话框的使用。

【预备知识】

1. 对话框 Dialog 类

Dialog 类和 Frame 类都是 Window 的子类。但不同之处是，对话框必须依赖于某个窗口或组件，当它所依赖的窗口或组件消失时，对话框也消失；当它所依赖的窗口或组件可见时，对话框会自动恢复。

创建对话框与创建窗口类似，通过继承一个 Dialog 类来建立一个对话框类。对话框也是一个容器，它的默认布局方式是 BorderLayout，可以添加组件，实现与用户的交互操作。

Dialog 类的构造方法见表 3.13。

表 3.13 Dialog 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| Dialog(Frame f,String s) | 构造一个具有标题 s 的初始不可见的对话框，f 是对话框所依赖的窗口 |
| Dialog(Frame f,String s,boolean b) | 构造一个具有标题 s 的初始不可见的对话框，f 是对话框所依赖的窗口，b 决定对话框是有模式或无模式 |

Dialog 对话框的模式分为无模式和有模式两种。

如果一个对话框是有模式的，那么当这个对话框处于激活状态时，只让程序响应对话框内部的事件，程序不能再激活它所依赖的窗口或组件，当完成操作关闭对话框后，方可对它所依赖的窗口进行操作。

无模式对话框处于激活状态，程序仍然能够激活它所依赖的窗口或组件。

Dialog 类的常用方法见表 3.14。

表 3.14 Dialog 类的常用方法

| 方法 | 说明 |
| --- | --- |
| getTitle() | 获取对话框的标题 |
| setTitle() | 设置对话框的标题 |
| setModal(boolean b) | 设置对话框的模式 |
| setSize() | 设置对话框的大小 |
| setVisible(boolean b) | 显示或隐藏对话框 |

2. 文件对话框 FileDialog 类

FileDialog 是 Dialog 的子类，它创建的对象称为文件对话框。文件对话框是一个打开文件和保

存文件的有模式对话框。文件对话框必须依赖一个 Frame 对象。

FileDialog 类的主要方法见表 3.15。

表 3.15　FileDialog 类的主要方法

| 方法 | 说明 |
| --- | --- |
| FileDialog(Frame f,String s,int mode) | 构造方法，f 为所依赖的窗口对象，s 是对话框的名称，mode 取值为 FileDialog.LOAD 或 FileDialog.SAVE |
| public String getDirectory() | 获取当前对话框中所显示的文件目录 |
| public String getFile() | 获取对话框中显示的文件的字符串表示，如不存在则为 null |

当文件对话框处于激活状态时，在“文件名”输入栏中输入文件名后，无论单击对话框中的“打开”按钮或“取消”按钮，对话框都自动消失，不能实现对文件的打开或保存操作，因为文件对话框仅仅提供了一个操作的界面，不能真正实现对文件的输入/输出操作。当单击了文件对话框上的“打开”或“保存”按钮后，getFile()方法才能返回非空的字符串对象。

3. 消息对话框 JOptionPane 类

消息对话框是有模式对话框，进行一个重要的操作动作之前，最好能弹出一个消息对话框进行确认。可以调用 Javax.swing 包中的 JOptionPane 类的静态方法创建：

```
public static void showMessageDialog(
    Component parentComponent,      //消息对话框依赖的组件
    String message,                 //要显示的消息
    String title,                   //对话框的标题
    int messageType);               //对话框的外观，取值如下：
JOptionPane.INFORMATION_MESSAGE
JOptionPane.WARNING_MESSAGE
JOptionPane.ERROR_MESSAGE
JOptionPane.QUESTION_MESSAGE
JOptionPane.PLAIN_MESSAGE
```

4. 确认对话框 JOptionPane 类

确认对话框是有模式对话框，使用 javax.swing 包中的 JOptionPane 类的静态方法创建：

```
public static int showConfirmDialog(
    Component parentComponent,      //对话框所依赖的组件
    Object mesage,                  //对话框上显示的消息
    String title,                   //对话框的标题
    int optionType);                //对话框的外观，取值如下：
JOptionPane.YES_NO_OPTION
JOptionPane.YES_NO_CANCEL_OPTION
JOptionPane.OK_CANCEL_OPTION
```

当对话框消失后，showConfirmDialog 方法会返回下列整数之一：

```
JOptionPane.YES_OPTION
JOptionPane.NO_OPTION
JOptionPane.CANCEL_OPTION
JOptionPane.OK_OPTION
JOptionPane.CLOSED_OPTION
```

返回的具体值依赖于用户单击了对话框上的哪个按钮以及对话框上的关闭图标。

5. 颜色对话框 JColorChooser 类

使用 javax.swing 包中的 JColorChooser 类的静态方法创建：

```
public static Color showDialog(
    Component component,            //对话框所依赖的组件
    String title,                   //对话框的标题
    Color initialColor);            //对话框消失后返回的默认颜色
```

该类可根据用户在颜色对话框中选择的颜色返回一个颜色对象。

## 【任务实施】

（1）定义“帮助”对话框类。

```
import java.awt.Button;
import java.awt.Dialog;
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

//定义"帮助"对话框
public class DHelp extends Dialog implements ActionListener{
    Label lmessage;
    Button bClose;
    Panel p;
    //构造方法
    DHelp(Frame f,String s,boolean b){
        //父类的构造方法
        super(f,s,b);
        lmessage=new Label("本程序由 11 软件技术班完成");
        bClose=new Button("关闭");
        p=new Panel();
        //设置布局方式为网格型
        GridLayout g=new GridLayout(2,1);
        this.setLayout(g);
        this.add(lmessage);
        this.add(p);
        p.add(bClose);

        bClose.addActionListener(this);
    }
    public void actionPerformed(ActionEvent arg0) {
        //设置对话框不可见
        this.setVisible(false);

    }
}
```

（2）修改 MainFrame 类，添加“帮助”菜单功能实现。

```
public void actionPerformed(ActionEvent e) {
//实现"帮助"菜单功能
    if(e.getSource()==miHelp){
        //实例化"帮助"对话框
        DHelp help=new DHelp(this,"帮助信息",false);
```

```
            //设置对话框大小
            help.setSize(200, 200);
            //设置对话框可见
            help.setVisible(true);
    }
```

（3）添加“颜色”菜单，其中包含“字体颜色”和“背景颜色”两个菜单项。

```
public void actionPerformed(ActionEvent e) {
    //实现"字体颜色"菜单功能
    if(e.getSource()==miColorFore){
        //通过颜色对话框获得用户选择的颜色存储在颜色对象中
        Color newColor=JColorChooser.showDialog(this,"调色板",Color.BLACK);
        //设置字体颜色为用户选择颜色
        ta.setForeground(newColor);

    }
    //实现"背景颜色"菜单功能
    if(e.getSource()==miColorBackground){
        //通过颜色对话框获得用户选择的颜色存储在颜色对象中
        Color newColor=JColorChooser.showDialog(this,"调色板",Color.BLACK);
        //设置背景颜色为用户选择颜色
        ta.setBackground(newColor);
    }
}
```

【任务小结】

通过本任务，实现自定义对话框及调用颜色对话框，完成“帮助”及“颜色”菜单功能。

【思考与习题】

1.（　　）类用于创建菜单项。

A．MenuItem　　B．PopupMenu　　C．Menu　　D．MenuBar

2．以下菜单类中，（　　）是父类。

A．CheckBoxMenuItem　　B．RadioButtonMenuItem

C．Menu　　D．MenuItem

3．编写一个应用程序，用户可以在一个文本框中输入数字字符，按 Enter 键后将数字放入一个文本区域。当输入的数字大于 1000 时，弹出一个有模式的对话框，提示用户数字已经大于 1000，是否继续将该数字放入文本区。

## 任务四　记事本的打开与保存功能

【任务描述】

实现记事本的文件打开和文件保存的功能。

【任务分析】

应用程序中的数据要写入到硬盘的某个文件上，或者要将硬盘的某个文件的内容读取到应用程

序中，都需要读或写某个文件。要实现文件读写，需要掌握输入输出流的操作。

本任务的关键点：

- 文本文件的读写操作。

【预备知识】

1. File 类

File 对象既可表示文件，也可表示目录。在程序中一个 File 对象可以代表一个文件或目录。利用它可对文件或目录及其属性进行基本操作。通过它可以获得与文件相关的信息，如名称、文件大小、最后修改日期等。File 类的继承关系如图 3-8 所示。

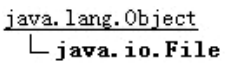


图 3-8 File 类的继承关系图

File 类的构造方法见表 3.16。

表 3.16 File 类的构造方法

| 构造方法 | 说明 |
|---|---|
| File(String filename) | 根据文件的路径创建 File 实例 |
| File(String directoryPath,String filename) | 根据文件路径和文件名创建 File 实例 |
| File(File f,String filename) | 根据文件的抽象路径名和文件路径创建一个 File 实例 |

File 类的常用方法见表 3.17。

表 3.17 File 类的常用方法

| 方法 | 说明 |
|---|---|
| String getAbsolutePath() | 返回此抽象路径名的绝对路径名字符串 |
| String getName() | 返回由此抽象路径名表示的文件或目录的名称。该名称是路径名名称序列中的最后一个名称。如果路径名名称序列为空，则返回空字符串 |
| String getParent() | 返回此抽象路径名父目录的路径名字符串；如果此路径名没有指定父目录，则返回 null |
| String getPath() | 将此抽象路径名转换为一个路径名字符串 |
| boolean isDirectory() | 测试此抽象路径名表示的文件是否是一个目录 |
| boolean isFile() | 测试此抽象路径名表示的文件是否是一个标准文件 |
| boolean isHidden() | 测试此抽象路径名指定的文件是否是一个隐藏文件 |
| long lastModified() | 返回此抽象路径名表示的文件最后一次被修改的时间 |
| long length() | 返回由此抽象路径名表示的文件的长度 |
| boolean exists() | 测试此抽象路径名表示的文件或目录是否存在 |

例 3.4 获取某个文件的一些信息。

```
public class example3_3{
    public static void main(String []args){
        File f=new File("d:\java\example\readme.txt");
```

```
            System.out.println("文件的名称是"+f.getName());
            System.out.println("文件是否存在"+f.exits());
            System.out.println("文件的长度"+f.length());
            System.out.println("文件的路径是"+f.getAbsolutePath());
            System.out.println("是否是文件"+f.isFile());
            System.out.println("是否是目录"+f.isDirectory());
            System.out.println("文件的最后修改时间"+f.lastModified());
        }
    }
```

2. 流

Java 中对文件的操作是以流的方式进行的。流是 Java 内存中一组有序数据序列。Java 将数据从源（文件、内存、键盘、网络）读入到内存中，形成了流，然后将这些流写到另外的目的地（文件、内存、控制台、网络）。之所以称为流，是因为这个数据序列在不同时刻所操作的是源的不同部分。

在 Java 类库中，IO 部分的内容很庞大，它涉及的领域很广泛：标准输入输出、文件的操作、网络上的数据流、字符串流、对象流、zip 文件流。

- 按流向分

输入流：程序可以从中读取数据的流。

输出流：程序能向其中写入数据的流。

- 按数据传输单位分

字节流：以字节为单位传输数据的流。

字符流：以字符为单位传输数据的流。

- 按功能分

节点流：用于直接操作目标设备的流。

过滤流：是对一个已存在的流的链接和封装，通过对数据进行处理为程序提供功能强大、灵活的读写功能。

JDK 所提供的所有流类位于 java.io 包中，分别继承自以下四种抽象流类。

- InputStream：继承自 InputStream 的流都是用于向程序中输入数据的，且数据单位都是字节（8 位）。
- OutputSteam：继承自 OutputStream 的流都是程序用于向外输出数据的，且数据单位都是字节（8 位）。
- Reader：继承自 Reader 的流都是用于向程序中输入数据的，且数据单位都是字符（16 位）。
- Writer：继承自 Writer 的流都是程序用于向外输出数据的，且数据单位都是字符（16 位）。

3. FileInputStream 类和 FileOutputStream 类

FileInputStream 从文件系统中的某个文件中获得输入字节，如果用户的文件读取需求比较简单，那么用户可以使用该类，该类是从 InputStream 中派生出来的简单输入类，该类的实例方法都是从 InputStream 类继承来的。其继承关系如图 3-9 所示。

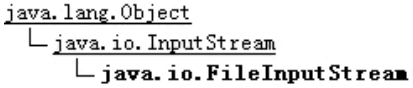


图 3-9　FileInputStream 类的继承关系图

FileInputStream 类的构造方法见表 3.18。

表 3.18 FileInputStream 类的构造方法

| 构造方法 | 说明 |
| --- | --- |
| FileInputStream(File file) | 通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的 File 对象 file 指定 |
| FileInputStream(String name) | 通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的路径名 name 指定 |

FileInputStream 类的常用方法见表 3.19。

表 3.19 FileInputStream 类的常用方法

| 方法 | 说明 |
| --- | --- |
| int available() | 返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取（或跳过）的估计剩余字节数 |
| void close() | 关闭此文件输入流并释放与此流有关的所有系统资源 |
| int read() | 从此输入流中读取一个数据字节 |
| int read(byte[] b) | 从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中 |
| int read(byte[] b,<br>int off,int len) | 从此输入流中将最多 len 个字节的数据读入一个 byte 数组中 |

使用 FileInputStream 类读取文本文件的步骤：

第一步：导入相关类

```
import java.io.IOException;
import java.io.InputStream;
import java.io.FileInputStream;
```

第二步：构造一个文件输入流对象

```
InputStream fileobject = new FileInputStream("text.txt");
```

第三步：利用文件输入流类的方法读取文本文件的数据

```
fileobject.available();      //可读取的字节数
fileobject.read();           //读取文件的数据
```

第四步：关闭文件输入流对象

```
fileobject.close();
```

**例 3.5** 使用 FileInputStream 类读取操作系统上的某个文件。

```
public class Example2_4{
public static void main(String[] args) throws IOException {
        int size;
        InputStream fileobject = new FileInputStream("t1.txt");
        System.out.println("可读取的字节数: "+ (size = fileobject.available()));
        char[] text = new char[200] ;
        for (int count = 0; count < size; count++) {
            text[count] = ((char) fileobject.read());
            System.out.print(text[count]);
        }
        System.out.println("");
        fileobject.close();
    }
```

FileOutputStream 类与 FileInputStream 类相对应，FileOutputStream 类称为文件输入流，它的作

用是把内存中的数据输出到文件中去。它是一个字节输出流 OutputStream 抽象类的一个子类。其继承关系如图 3-10 所示。

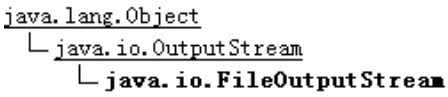


图 3-10　FileOutputStream 类的继承关系图

FileOutputStream 类的构造方法见表 3.20。

表 3.20　FileOutputStream 类的构造方法

| 构造方法 | 说明 |
|---|---|
| FileOutputStream(File file) | 创建一个向指定 File 对象表示的文件中写入数据的文件输出流 |
| FileOutputStream(String name) | 创建一个向具有指定名称的文件中写入数据的输出文件流 |

FileOutputStream 类的常用方法见表 3.21。

表 3.21　FileOutputStream 类的常用方法

| 方法 | 说明 |
|---|---|
| void write(int b) | 将指定字节写入此文件输出流 |
| void write(byte[] b) | 将 b.length 个字节从指定 byte 数组写入此文件输出流中 |
| void write(byte[] b,<br>int off,int len) | 将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此文件输出流 |
| void close() | 关闭此文件输出流并释放与此流有关的所有系统资源 |

使用 FileOutputStream 类写进文本文件的步骤：

第一步：导入相关类

```
import java.io.IOException;
import java.io.OutputStream;
import java.io.FileOutputStream;
```

第二步：构造一个文件输出流对象

```
OutputStream fos = new FileOutputStream("Text.txt");
```

第三步：利用文件输出流的方法写文本文件

```
String str ="好好学习 Java";
byte[] words = str.getBytes();
fos.write(words, 0, words.length);
```

第四步：关闭文件输出流

```
fos.close();
```

**例 3.6**　使用 FileOutputStream 类将一个字符串写入文本文件。

```
public class Example3_5{
    public static void main(String[] args) {
        try {
            String str ="好好学习 Java";
            byte[] words    = str.getBytes();
            OutputStream fos = new FileOutputStream("Text.txt");
            fos.write(words, 0, words.length);
            System.out.println("Text 文件已更新!");
```

```
                fos.close();
            } catch (IOException obj) {
                    System.out.println("创建文件时出错!");
        }
    }
```

4. BufferedReader 类和 BufferedWriter 类

BufferedReader 类可以从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取。可以指定缓冲区的大小，或者可使用默认的大小。大多数情况下，默认值就足够大了。其继承关系如图 3-11 所示。

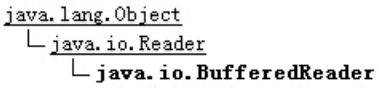


图 3-11　BufferedReader 类的继承关系图

BufferedReader 类的构造方法见表 3.22。

表 3.22　BufferedRreader 类的构造方法

| 构造方法 | 说明 |
|---|---|
| BufferedReader(Reader in) | 创建一个使用默认大小输入缓冲区的缓冲字符输入流 |
| BufferedReader(Reader in, int sz) | 创建一个使用指定大小输入缓冲区的缓冲字符输入流 |

BufferedReader 类的常用方法见表 3.23。

表 3.23　BufferedReader 类的常用方法

| 方法 | 说明 |
|---|---|
| int read() | 读取单个字符 |
| int read(char[] cbuf,int off,int len) | 将字符读入数组的某一部分 |
| String readLine() | 读取一个文本行 |
| void close() | 关闭该流并释放与之关联的所有资源 |

使用 BufferedReader 类从文本文件中读取字符的步骤：

第一步：引入相关的类

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
```

第二步：构造一个 BufferedReader 对象

```
FileReader fr=new FileReader("mytest.txt");
BufferedReader br=new BufferedReader(fr);
```

第三步：读取文本文件的数据

```
br.readLine();          //读取一行数据，返回字符串
```

第四步：关闭相关的流对象

```
br.close();
fr.close();
```

**例 3.7**　使用 BufferedReader 类读取文本文件。

```
public class Example3_6{
    public static void main(String []args){
```

```
            /**创建一个 FileReader 对象.*/
            FileReader fr=new FileReader("mytest.txt");
            /**创建一个 BufferedReader  对象.*/
            BufferedReader br=new BufferedReader(fr);
            /**读取一行数据.*/
            String line=br.readLine();
            while(line!=null){
                System.out.println(line);
                line=br.readLine();
            }
        }
    }
```

BufferedWriter 类可以以字符流的方式通过缓冲区把数据写入文本文件，可以提高写入文本文件的效率。BufferedWriter、FileWriter 都是字符输出流 Writer 抽象类下的子类。其继承关系如图 3-12 所示。

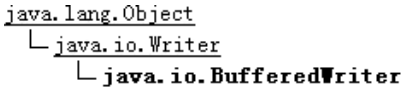


图 3-12　BufferedWriter 类的继承关系图

BufferedWriter 类的构造方法见表 3.24。

表 3.24　BufferedWriter 类的构造方法

| 构造方法 | 说明 |
|---|---|
| BufferedWriter(Writer out) | 创建一个使用默认大小输出缓冲区的缓冲字符输出流 |
| BufferedWriter(Writer out, int sz) | 创建一个使用给定大小输出缓冲区的新缓冲字符输出流 |

BufferedWriter 类的常用方法见表 3.25。

表 3.25　BufferedWriter 类的常用方法

| 方法 | 说明 |
|---|---|
| void write(int c) | 写入单个字符 |
| void write(char[] cbuf,int off,int len) | 写入字符数组的某一部分 |
| void write(String s,int off,int len) | 写入字符串的某一部分 |
| void newLine() | 写入一个行分隔符 |
| void close() | 关闭此流，但要先刷新它 |

使用 BufferedWriter 类写文本文件的步骤：

第一步：引入相关的类

```
import java.io.FileWriter ;
import java.io.BufferedWriter ;
import java.io.IOException;
```

第二步：构造一个 BufferedWriter 对象

```
FileWriter fw=new FileWriter("mytest.txt");
BufferedWriter bw=new BufferedWriter(fw);
```

第三步：利用 BufferedWriter 的方法写文本文件

```
bw.write ("hello");
```

第四步：相关流对象的清空和关闭

```
bw.flush();
fw.close();
```

例 3.8　将一个字符串写入文本文件。

```
public class Example3_7{
    public static void main(String []args){
        /**创建一个 FileWriter 对象*/
        FileWriter fw=new FileWriter("mytest.txt");
        /**创建一个 BufferedWriter 对象*/
        BufferedWriter bw=new BufferedWriter(fw);
        bw.write("大家好！");
        bw.write("我正在学习 BufferedWriter。");
        bw.newLine();
        bw.write("请多多指教！");
        bw.newLine();
        bw.write("email: study@sina.com.cn");
        bw.flush();
        fw.close();
    }
}
```

例 3.9　将 C 盘根目录下的 setup.txt 文件内容复制到 D 盘根目录下的 copy.txt 文件中。

```
public class Example3_8{
        public static void main(String []args){
        FileReader fis = new FileReader ("c:/setup.txt");
        BufferedReader dis = new BufferedReader (fis);
        FileWriter outFile = new FileWriter ("d:/copy.txt");
        BufferedWriter out = new BufferedWriter (outFile);
        String temp;
        while ( (temp = dis.read()) != -1) {
            out.write(temp);
        }
        fis.close();
        dis.close();
        outFile.close();
        out.close();
    }
}
```

## 【任务实施】

（1）在构造方法中添加如下代码：

```
String currentFileName;
void saveAsFile(){
        filedialog_save=new FileDialog(this,"保存",FileDialog.SAVE);
        filedialog_save.setVisible(true);
        String dir=filedialog_save.getDirectory();
        String name=filedialog_save.getFile();
        currentFileName=dir+name;
        saveFile();

    }
```

```
    void saveFile(){
        if(currentFileName==null){
            saveAsFile();
        }else{
            File file=new File(currentFileName);
            try{
                FileWriter fw=new FileWriter(file);
                BufferedWriter out=new BufferedWriter(fw);
                out.write(t.getText());
                out.flush();
                fw.close();
            }catch(Exception eo){
                eo.printStackTrace();
            }
        }
    }
```

（2）在事件的处理方法中添加如下代码：

```
public void actionPerformed(ActionEvent e){
if(e.getSource()==fileOpen){
        filedialog_open=new FileDialog(this,"打开文件对话框",FileDialog.LOAD);
        filedialog_open.setVisible(true);
        File file;
        String dir=filedialog_open.getDirectory();
        String name=filedialog_open.getFile();
        file=new File(dir+name);
        try{
            FileReader fr=new FileReader(file);
            BufferedReader in=new BufferedReader(fr);
            String line=in.readLine();
            String text=line+"\n";
            while(line!=null){

                text=text+line+"\n";
                line=in.readLine();
            }
                t.setText(text);
                in.close();
                fr.close();
            }catch(Exception ie){
                ie.printStackTrace();
            }
        }
    if(e.getSource()==fileSave){
        saveFile();
    }
    if(e.getSource()==fileSaveAs){
        saveAsFile();
    }
}
```

【任务小结】

通过本任务实现记事本的“打开”、“保存”和“另存为”的功能。若要读取文本文件需要掌握java.io 包中的相关输入输出流类，通过它们的方法实现读或写。

【思考与习题】

1．File 类中的（　　）方法可以用来判断文件或目录是否存在。

A．exist()　　B．exists()

C．fileExist()　　D．fileExists()

2．File 类中的（　　）方法可以用来获取文件的大小。

A．length()　　B．size()

C．getLength()　　D．getSize()

3．文本文件的读写过程中，需要处理以下（　　）异常。

A．ClassNotFoundException　　B．IOException

C．SQLException　　D．RemoteException

4．字符流是以（　　）传输数据的。

A．1 个字节　　B．8 位字符

C．16 位 Unicode 字符　　D．1 比特

# 任务五　打包程序

【任务描述】

将“记事本”应用程序打包，可以双击运行。

【任务分析】

一个 Java 应用程序开发调试完成后，需要将其压缩成一个 JAR 文件。这样可以脱离开发平台，直接使用 Java 解释器执行这个压缩文件。

本任务的关键点：

- JAR 文件的作用。
- 如何打包 JAR 文件。

【预备知识】

1．将应用程序压缩为 JAR 文件

假设应用程序中有两个类 A 和 B，其中 main()方法在 A 类中。生成一个 JAR 文件的步骤如下：

（1）首先用文件编辑器（如 Windows 下的记事本）编写一个清单文件：

**Mymoon.mf**

Manifest-Version: 1.0

Main-Class: A

保存 Mymoon.mf 到 D:\test。需要注意的是在编写清单文件时，在“Manifest-Version:”和“1.0”之间，“Main-Class:”和主类“A”之间必须有且只有一个空格。

（2）生成 JAR 文件

在命令控制台中输入：

D:\test\jar cfm Tom.jar Mymoon.mf A.class B.class

其中参数 c 表示要生成一个新的 JAR 文件；f 表示要生成的 JAR 文件的名字；m 表示文件清单文件的名字。

（3）执行 JAR 文件

在命令控制台中输入：

java -jar tom.jar

即可运行该应用程序。

也可以将 tom.jar 文件复制到任何一个安装了 Java 运行环境的计算机上，双击该文件就可以运行该 Java 应用程序了。

**注意**：*若计算机上安装了压缩程序 WinRAR，双击 JAR 文件，可能会打开 WinRAR 程序进行解压缩操作。解决方法：在 WinRAR 软件的设置中，取消对 JAR 扩展名的关联。*

除了手动方式外，可以在 MyEclipse 中导出 JAR 包，操作步骤如下：

（1）单击“File”菜单→“Export”，如图 3-13 所示。

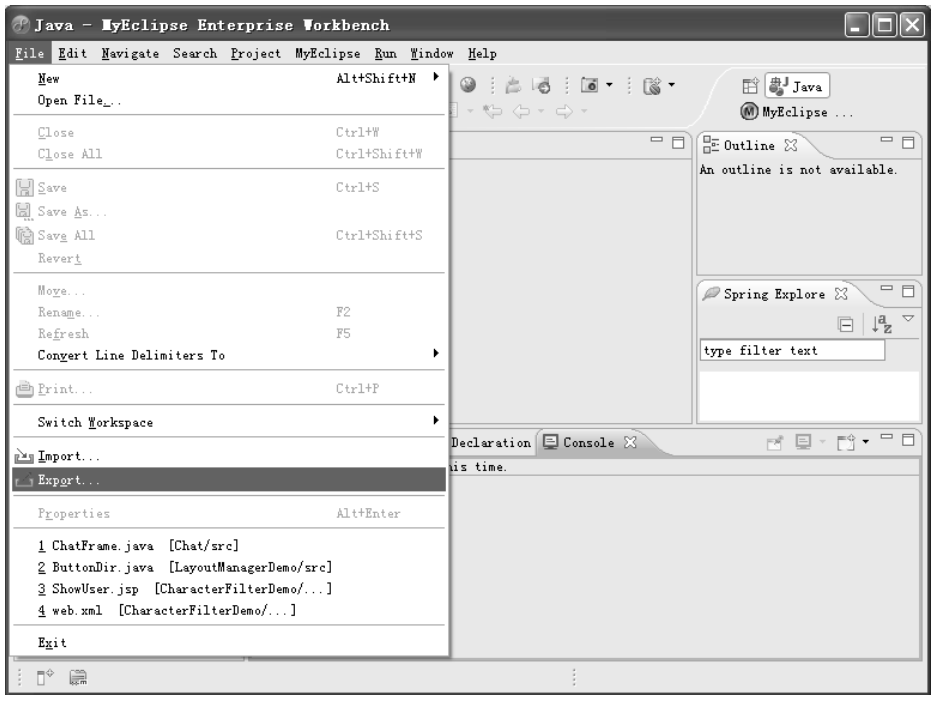


图 3-13　选择“Export”菜单命令

（2）打开“Export”窗口，如图 3-14 所示。在“Export”窗口中，选择“JAR file”类型。单击“Next”按钮。

（3）在“JAR Export”对话框中，左上部分选择需要导出的工程，右上部分显示该工程内包

含的文件，如图 3-15 所示。在“JAR file”文本框后，单击“Browse”按钮，弹出如图 3-16 所示对话框。

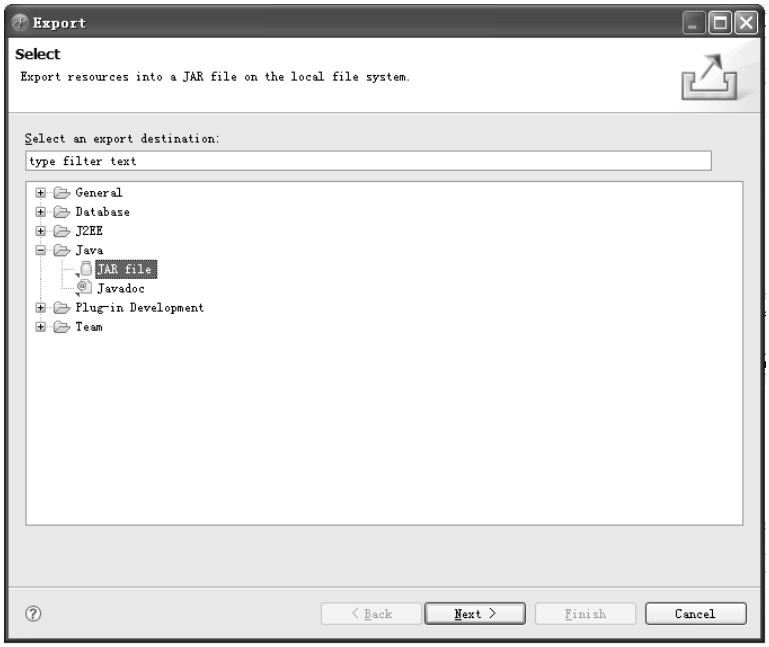


图 3-14　选择导出“JAR file”

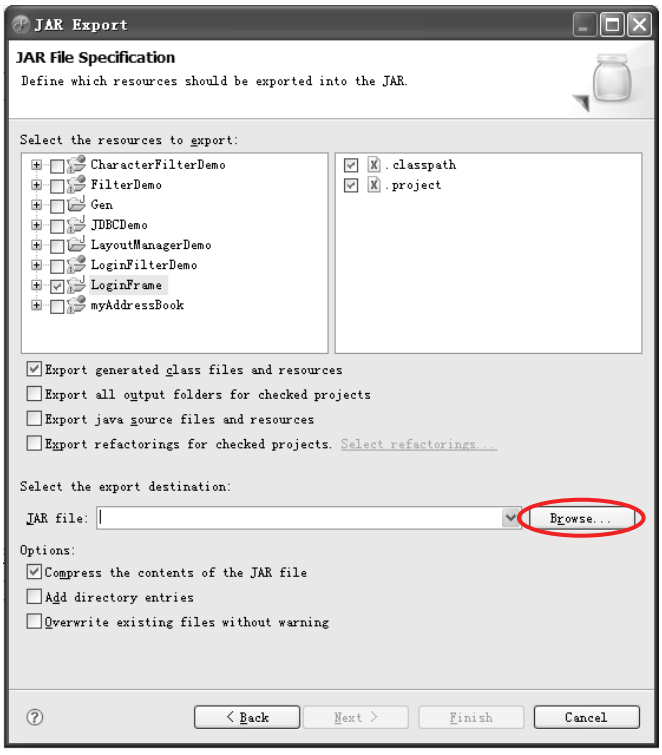


图 3-15　选择需要导出的工程及保存路径

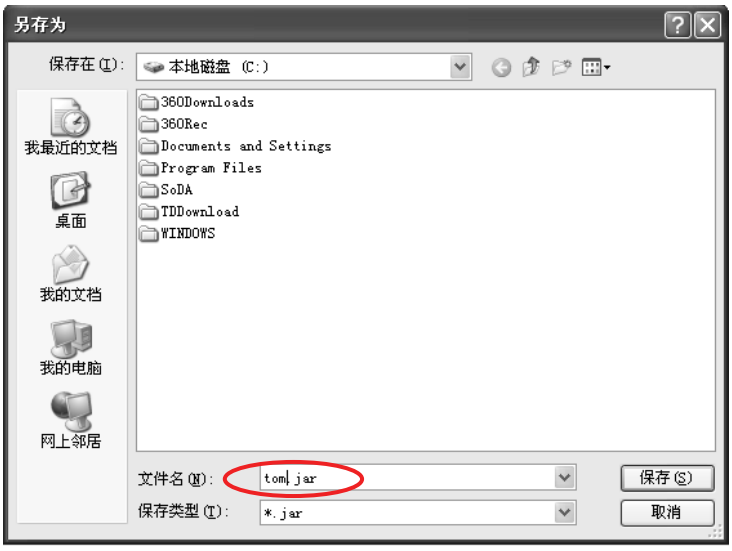


图 3-16　选择 JAR 文件保存路径

（4）在“另存为”对话框中，选择 JAR 文件的保存路径，并输入文件名称，扩展名为.jar。单击“保存”按钮，回到图 3-15 所示窗口，单击“Next”按钮。

（5）弹出如图 3-17 所示对话框，单击“Next”按钮。

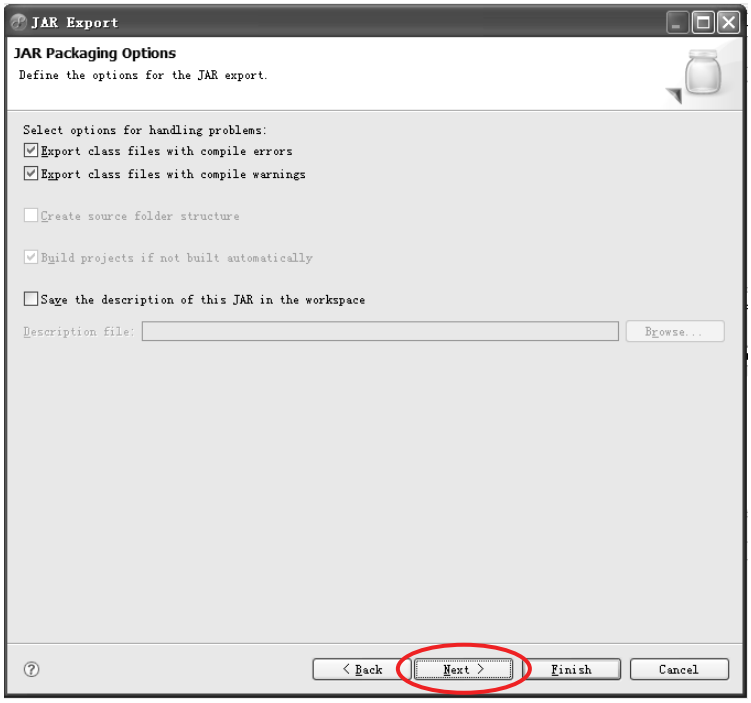


图 3-17　JAR 包选项窗口

（6）弹出如图 3-18 所示对话框，在“Main class”文本框后，单击“Browse”按钮，弹出如图 3-19 所示对话框。在该对话框中，选择本项目中 Main 方法所在的类。单击“OK”按钮。

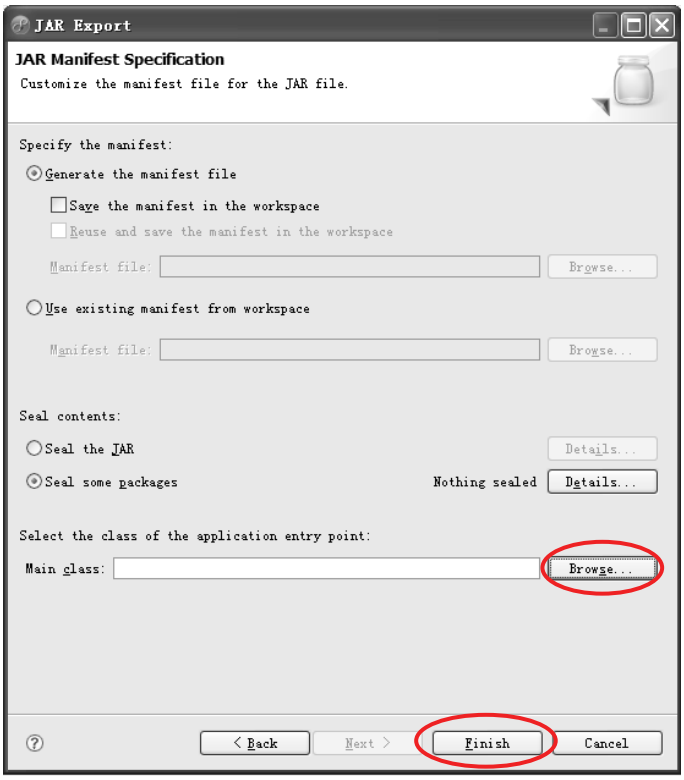


图 3-18　设置 Main class


图 3-19　选择 Main Class

（7）在“JAR Export”对话框中，在“Main class”文本框内，显示该类的名称。单击“Finish”按钮，完成导出。如图 3-20 所示。

打开所选路径 C:\，如图 3-21 所示，找到 tom.jar 文件，双击该文件，即可运行该应用程序。

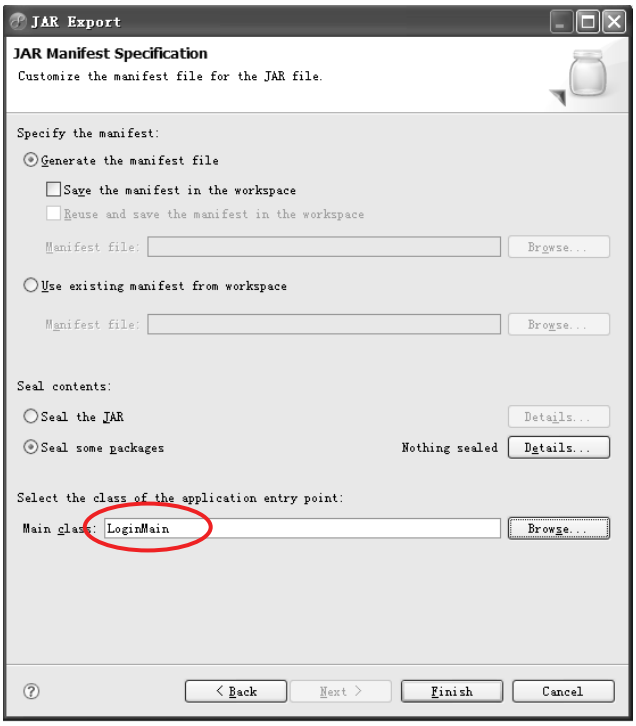


图 3-20　设置完成 Main Class


图 3-21　完成导出 JAR 文件

2. 将类压缩成 JAR 文件

可以使用 jar.exe 把一些类的字节码文件压缩成一个 JAR 文件，然后将这个 JAR 文件存放到 Java 运行环境的扩展中，即将该 JAR 文件存放在 JDK 安装目录的 jre\lib\ext 文件夹中。这样，其他的程

序就可以使用这个 JAR 文件中的类来创建对象了。

（1）编写一个清单文件（Manifestfiles）

**moon.mf**

Manifest-Version: 3.0

Class: Test1 Test2

保存 moon.mf 到 D:\test

（2）生成 JAR 文件

D:\test\jar cfm Jerry.jar moon.mf Test1.class Test2.class

（3）导入 JAR 文件

该 JAR 文件存放在 JDK 安装目录的 jre\lib\ext 文件夹中。

也可以使用 MyEclipse 将类压缩成 JAR 文件。其过程与将工程压缩成 JAR 文件是一样的，只是不需要制定 main 方法所在的类。

### 【任务实施】

按照下列步骤完成“记事本”程序的压缩：

（1）首先用文件编辑器（如 Windows 下的记事本）编写一个清单文件：

**MANIFEST.MF**

Manifest-Version: 1.0

Main-Class: Test

保存 MANIFEST.MF 到 D:\test。

（2）生成 JAR 文件

在命令控制台中输入：

D:\test\jar cfm Editor.jar MANIFEST.MF Test.class MainFrame.class DHelp.class

（3）执行 JAR 文件

在命令控制台中输入：

java -jar Editor.jar

即可运行该应用程序，或直接双击该 JAR 包。

### 【任务小结】

通过该任务，了解 JAR 包的作用，掌握如何使一个开发完成的 Java 应用程序脱离开发平台，独立运行。

### 【思考与习题】

使用 MyEclipse 将“记事本”程序压缩成 JAR 包。

## 请记住以下英语单词

Menu ['menju:] 菜单

bar [bɑ:] 条，块

item ['aitəm] 一项，项目

separator ['sepə,reɪtə] 分离器

enable [i'neibl] 使能够；使可能
compare [kəm'pɛə] 相比
length [leŋθ] 长，长度；距离
split [split] （使）裂开；（使）破裂
end [end] 结束；终止
append [ə'pend] 附加；添加
select [si'lekt] 选择；挑选
action ['ækʃən] 行动，活动
position [pə'ziʃən] 方位，位置
absolute ['æbsəlju:t] 绝对的，完全的
directory [di'rektəri] 目录
exist [ig'zist] 存在，有
read [ri:d] 读；看懂，理解
close [kləuz] （使）关，关闭；终止
save [seiv] 储蓄，贮存
export [eks'pɔ:t] 出口，输出
add [æd] 加，加入；增加，添加
index ['indeks] 索引
replace [ri'pleis] 更换，替换
start [stɑ:t] 开始；出发
value ['vælju] 价值，价格
insert [in'sə:t] 插入，嵌入
listener ['lisnə] 倾听者，收听者
source [sɔ:s] 来源，出处
file [fail] 卷宗，文件
path [pɑ:θ] 小路，小径
hidden ['hɪdn] 难以发现的，隐藏的
stream [stri:m] 流，一股，一串
write [rait] 写；写信
count [kaunt] 计数，总数
load [ləud] 装载，装载量
next [nekst] 接下去；然后