

## 第2章 第一个 Java 应用

任何一个软件应用都是对某种现实系统的模拟，在现实系统中会包含一个或多个实体，这些实体具有特定的属性和行为。本章介绍的 Java 应用所模拟的现实系统中包含 5 个实体：福娃贝贝、晶晶、欢欢、迎迎和妮妮，它们是中国首次举办的奥运会的吉祥物，如图 2-1 所示。这些娃娃都有名字属性，此外还能说话，说出自己的名字。



图 2-1 会说话的福娃

本章范例名为 dollapp 应用，它包括一个 Doll 类和 AppMain 类，其中 Doll 类代表福娃，Doll 类有个 speak() 方法，它模拟福娃的说话行为。AppMain 类是 dollapp 应用的主程序类，它提供了一个程序入口方法 main()，Java 虚拟机从这个 main() 方法开始运行 dollapp 应用。

### 2.1 创建 Java 源文件

Java 应用由一个或多个扩展名为“.java”的文件构成，这些文件被称为 Java 源文件，从编译的角度，则被称为编译单元（Compilation Unit）。本章的例子包含两个 Java 源文件：Doll.java 和 AppMain.java，例程 2-1 和例程 2-2 分别是它们的程序代码。

例程 2-1 Doll.java

```
public class Doll{
    /** 福娃的名字 */
    private String name;

    /** 构造方法 */
    public Doll(String name){
        this.name=name;           //设置福娃的名字
    }

    /** 说话 */
    public void speak(){
        System.out.println(name); //打印名字
    }
}
```

```

    }
}

```

例程 2-2 AppMain.java

```

public class AppMain{
    public static void main(String args[]){
        Doll beibei=new Doll("贝贝");    //创建福娃贝贝
        Doll jingjing=new Doll("晶晶");    //创建福娃晶晶
        Doll huanhuan=new Doll("欢欢");    //创建福娃欢欢
        Doll yingying=new Doll("迎迎");    //创建福娃迎迎
        Doll nini=new Doll("妮妮");        //创建福娃妮妮

        beibei.speak();                    //福娃贝贝说话
        jingjing.speak();                  //福娃晶晶说话
        huanhuan.speak();                  //福娃欢欢说话
        yingying.speak();                  //福娃迎迎说话
        nini.speak();                      //福娃妮妮说话
    }
}

```

在 Doll.java 文件中定义了一个 Doll 类，它有一个 name 属性和一个 speak()方法。Doll.java 文件由以下内容构成：

（1）类的声明语句：

```
public class Doll{ ... }
```

以上代码指明类的名字为“Doll”，public 修饰符意味着这个类可以被公开访问。

（2）类的属性（也称为成员变量）的声明语句：

```
private String name;
```

以上代码表明 Doll 类有一个 name 属性，字符串类型，private 修饰符意味着这个属性不能被公开访问。

（3）方法的声明语句和方法主体：

```

public void speak(){
    System.out.println(name); //打印名字
}

```

以上代码表明 Doll 类有一个 speak()方法，不带参数，返回类型为 void，void 表示没有返回值，public 修饰符意味着这个方法可以被公开访问。speak()后面紧跟的大括号为方法主体，代表 speak()方法的具体实现。在本例中，speak()方法打印福娃本身的名字，用于模拟福娃的说话行为。

在 AppMain.java 文件中定义了一个 main()方法，这是 Java 应用程序的入口方法，当运行 dollapp 应用时，Java 虚拟机将从 AppMain 类的 main()方法中的程序代码开始运行。在本例中，main()方法先用 new 语句创建 5 个 Doll 对象，接着依次调用它们的 speak()方法。

### 2.1.1 Java 源文件结构

一个Java应用包含一个或多个Java源文件，每个Java源文件只能包含下列内容（空格和注释除外）：

- l 零个或一个包声明语句（Package Statement）。
- l 零个或多个包引入语句（Import Statement）。
- l 零个或多个类的声明（Class Declaration）。
- l 零个或多个接口声明（Interface Declaration）。

每个Java源文件可包含多个类或接口的定义，但是至多只有一个类或者接口是public的，而且Java源文件必须以其中public类型的类的名字命名。例如，在以下例程2-3的AppMain.java中同时声明了AppMain类和Doll类，只有AppMain类被public修饰，并且该文件以AppMain类的名字命名，如果把AppMain.java文件改名为Doll.java，那么编译时会出现错误。

例程 2-3 AppMain.java

```
public class AppMain{...}  
class Doll{...}
```

### 2.1.2 包声明语句

包声明语句用于把Java类放到特定的包中，例如在例程2-4的AppMain.java中，AppMain类和Doll类都位于com.abc.dollapp包中。

例程 2-4 AppMain.java

```
package com.abc.dollapp;  
public class AppMain{...}  
class Doll{...}
```

在一个Java源文件中，最多只能有一个package语句，但package语句不是必需的。如果没有提供package语句，就表明Java类位于默认包中，默认包没有名字。

package语句必须位于Java源文件的第一行（忽略注释行）。以下3段代码分别表示AppMain.java的源代码，其中第一段和第二段是合法的，而第三段会导致编译错误。

第一段代码：

```
/** 注释行 */  
package com.abc.dollapp;  
public class AppMain{...}  
class Doll{...}
```

第二段代码：

```
package com.abc.dollapp;  
public class AppMain{...}  
class Doll{...}
```

第三段代码：

```
public class AppMain{...}
class Doll{...}
package com.abc.dollapp;
```

## Tips

在一个 Java 源文件中只允许有一个 package 语句，因此，在同一个 Java 源文件中定义的多个 Java 类或接口都位于同一个包中。

### 1. 包的作用

把类放到特定的包中，有三大作用：

(1) 能够区分名字相同的类。例如有两个类的名字均为 **Book**，其中一个表示书店的书，一个表示旅馆的订单，如何区分这两个类呢？只要把它们放到不同的包中，就相当于为它们指定了不同的名字空间。比如把代表书的 **Book** 类放到 **com.abc.bookstore** 包中，把代表订单的 **Book** 类放到 **com.abc.hotel** 包中，这样，**com.abc.bookstore.Book** 和 **com.abc.hotel.Book** 分别代表不同的 **Book** 类。

## Tips

在本书中，如果类名中包括类所在的包的信息，这种类名称为完整类名，或者称为完整限定名，例如 **com.abc.bookstore.Book** 就是完整限定名。如果类位于默认包中，那么它的完整限定名就是类名。

(2) 有助于实施访问权限控制。当位于不同包之间的类相互访问时，会受到访问权限的约束，参见本书第 7 章的 7.1 节（访问控制修饰符）。

(3) 有助于划分和组织 Java 应用中的各个类。假定 ABC 公司开发了一个购物网站系统，名为 **netstore** 应用，在这个应用中，共有 300 个类，其中有一部分类位于客户端，用于构建客户端界面，有一部分类位于服务器端，用于处理业务逻辑，还有一些是公共类，提供了各种实用方法。可以把 **netstore** 应用分为 3 个顶层包：

- ┆ **com.abc.netstore.client**：这个包中的类用于构建客户端界面。
- ┆ **com.abc.netstore.server**：这个包中的类用于处理业务逻辑。
- ┆ **com.abc.netstore.common**：这个包中的类提供了各种实用方法。

对于位于服务器端的类，有一部分类负责管理订单信息，有一部分类负责管理库存信息，有一部分类负责管理客户信息，因此把 **com.abc.netstore.server** 包又分为 3 个子包：

- ┆ **com.abc.netstore.server.order**：这个包中的类负责管理订单信息。
- ┆ **com.abc.netstore.server.store**：这个包中的类负责管理库存信息。
- ┆ **com.abc.netstore.server.customer**：这个包中的类负责管理客户信息。

对于实际项目，到底如何划分包的结构，并没有统一的模式，开发者可根据实际情况来灵活地划分包的结构。

### 2. 包的命名规范

包的名字通常采用小写，包名中包含以下信息：

- ┆ 类的创建者或拥有者的信息。
- ┆ 类所属的软件项目的信息。

类在具体软件项目中所处的位置。

例如,假定有一个 SysContent 类的完整类名为 com.abc.netstore.common.SysContent 类,从这个完整类名中可以看出, SysContent 类由 ABC 公司开发,属于 netstore 项目,位于 netstore 项目的 common 包中。

包的命名规范实际上采用了 Internet 网上 URL 命名规范的反转形式。例如在 Internet 网上网址的常见形式为: http://netstore.abc.com, 而 Java 包名的形式则为: com.abc.netstore。

值得注意的是,Java 语法规则并不强迫包名必须符合以上规范。不过,以上命名规范能帮助应用程序确立良好的编程风格。

### 3. JDK 提供的 Java 基本包

JDK 提供了一些 Java 基本包,主要包括:

- l java.lang 包: 包含线程类 (Thread)、异常类 (Exception)、系统类 (System)、整数类 (Integer) 和字符串类 (String) 等,这些类是编写 Java 程序经常用到的。这个包是 Java 虚拟机自动引入的,也就是说,即使程序中没用提供“import java.lang.\*”语句,这个包也会被自动引入。
- l java.awt: 抽象窗口工具箱包,AWT 是“Abstract Window Toolkit”的缩写,这个包中包含用于构建 GUI 界面的类及绘图类。
- l java.io 包: 输入/输出包,包含各种输入流类和输出流类,如文件输入流类 (FileInputStream 类),以及文件输出流类 (FileOutputStream) 等。
- l java.util 包: 提供一些实用类,如日期类 (Date) 和集合类 (Collection) 等。
- l java.net 包: 支持 TCP/IP 网络协议,包含 Socket 类,以及和 URL 相关的类,这些类都用于网络编程。

除了上面提到的基本包,JDK 中还有很多其他的包,比如用于数据库编程的 java.sql 包,用于编写网络程序的 java.rmi 包 (RMI 是“Remote Method Invocation”的缩写)。另外, javax.\* 包是对基本包的扩展,包括用于编写 GUI 程序的 javax.swing 包,以及用于编写声音程序的 javax.sound 包等。

JDK 的所有包中的类构成了 Java 类库,或者叫作 JavaSE API。用户创建的 Java 应用程序都依赖于 JavaSE API。例如,在 dollapp 应用中,Doll 类用到了 java.lang.System 类和 java.lang.String 类。由于 java.lang 包是被自动引入的,所以在 Doll 类中没有提供“import java.lang.\*”语句。

#### 2.1.3 包引入语句

如果一个类访问了来自另一个包 (java.lang 包除外) 中的类,那么前者必须通过 import 语句把这个类引入。例如,假定 AppMain 类和 Doll 类分别位于不同的包中,其中 Doll 类位于 com.abc.dollapp.doll 包中,参见例程 2-5,而 AppMain 类位于 com.abc.dollapp.main 包中,参见例程 2-6。由于 AppMain 类的主方法 main() 会访问 Doll 类,因此,AppMain 类通过 import 语句引入 Doll 类:

```
import com.abc.dollapp.doll.Doll;
```

以上代码指明引入 `com.abc.dollapp.doll` 包中的 `Doll` 类。以下代码则表明引入 `com.abc.dollapp.doll` 包中所有的类：

```
import com.abc.dollapp.doll.*;
```

假如程序仅需要访问 `com.abc.dollapp.doll` 包中的 `Doll` 类，那么以上两条 `import` 语句都能完成相同的功能，但是第一条 `import` 语句的性能更优，因为假使程序中有多个 `import` 语句，如果采用以下方式：

```
import com.abc.dollapp.main.*;
import com.abc.dollapp.doll.*;
```

Java 编译器必须搜索所有的包，来判断程序中的 `Doll` 类到底位于哪个包中，而如果采用以下方式：

```
import com.abc.dollapp.main.*;
import com.abc.dollapp.doll.Doll;
```

Java 编译器能够明确地知道 `Doll` 类位于 `com.abc.dollapp.doll` 包中。

## Tips

`import` 语句不会导致类的初始化。例如“`import java.util.*`”语句并不意味着 Java 虚拟机会把“`java.util`”包中的所有类加载到内存中并对它们初始化。类的初始化的概念参见本书第 10 章（类的生命周期）。

例程 2-5 Doll.java

```
package com.abc.dollapp.doll;
public class Doll{ ... }
```

例程 2-6 AppMain.java

```
package com.abc.dollapp.main;
import com.abc.dollapp.doll.Doll;
public class AppMain{
    public static void main(String args[]){
        Doll doll=new Doll();
        doll.speak();
    }
}
```

例程 2-6 的 `AppMain` 类的 `main()` 方法访问了 `Doll` 类，`Doll` 类的完整类名为 `com.abc.dollapp.doll.Doll`。关于包的引入，有以下值得注意的问题：

（1）如果一个类同时引用了两个来自于不同包的同名类，在程序中必须通过类的完整类名来区分这两个类。例如在下面例程 2-7 的 `AppMain` 类中，同时引用了 `com.abc.bookstore.Book` 类和 `com.abc.hotel.Book` 类。

例程 2-7 AppMain.java

```
package com.abc.dollapp.main;
import com.abc.bookstore.Book;
import com.abc.hotel.Book;
```

```
public class AppMain{
    public static void main(String args[]){
        com.abc.bookstore.Book book1=new com.abc.bookstore.Book ();
        com.abc.hotel.Book book2=new com.abc.hotel.Book ();
    }
}
```

(2) 尽管包名中的符号“.”能够体现各个包之间的层次结构，但是每个包都是独立的，顶层包不会包含子包中的类。例如以下 import 语句引入 com.abc 包中的所有类：

```
import com.abc.*;
```

以上 import 语句会不会把 com.abc.dollapp 包及 com.abc.dollapp.main 包中所有的类都引入呢？答案是否定的。如果希望同时引入这 3 个包中的类，必须采用以下方式：

```
import com.abc.*;
import com.abc.dollapp.*;
import com.abc.dollapp.main.*;
```

(3) package 和 import 语句的顺序是固定的，在 Java 源文件中，package 语句必须位于第一行（忽略注释行），其次是 import 语句，接着是类或接口的声明，以下程序代码是合法的：

```
package com.abc.dollapp.doll.*;
import com.abc.*;
import com.abc.dollapp.*;
import com.abc.dollapp.main.*;
public class Doll{ ... }
```

或者：

```
//这是一行注释
package com.abc.dollapp.doll.*;
import com.abc.*;
import com.abc.dollapp.*;
import com.abc.dollapp.main.*;
public class Doll{ ... }
```

### 2.1.4 方法的声明

在 Java 语言中，每个方法都属于特定的类，方法的声明必须位于类的声明之中，这是与 C 语言的不同之处。声明方法的格式为：

```
返回值类型 方法名（参数列表）{
    方法主体
}
```

方法名是任意合法的标识符。返回值类型是方法的返回数据的类型，如果返回值类型为 void，表示没有返回值。参数列表可包含零个或多个参数，参数之间以逗号“，”分开。以下是合法的方法声明：

```
void speak(){ //参数列表为空；没有返回值
    System.out.println(name);
}
```

```

    }

    void speak(String word1, String word2){ //参数列表中包含两个参数；没有返回值
        if(word1==null && word2==null)
            return; //结束本方法的执行
        if(word1!=null)
            System.out.println(word1);
        if(word2!=null)
            System.out.println(word2);
    }

    String getName(){ //返回值为 String 类型
        return name; //返回特定数据
    }

```

如果方法的返回类型是 `void`，那么方法主体中可以没有 `return` 语句，如果有 `return` 语句，那么该 `return` 语句不允许返回数据；如果方法的返回类型不是 `void`，那么方法主体中必须包含 `return` 语句，而且 `return` 语句必须返回相应类型的数据。`return` 语句有两个作用：

（1）结束执行本方法。例如对于 `speak(String word1, String word2)` 方法，如果方法调用者传递的参数 `word1` 和 `word2` 都是 `null`，就立即结束本方法的执行：

```

    if(word1==null && word2==null)
        return; //结束本方法的执行

```

（2）向本方法的调用者返回数据。

### 2.1.5 程序入口 `main()` 方法的声明

`main()` 方法是 Java 应用程序的入口点，每个 Java 应用程序都是从 `main()` 方法开始运行的。作为程序入口的 `main()` 方法必须同时符合以下 4 个条件：

- l 访问限制：`public`。
- l 静态方法：`static`。
- l 参数限制：`main(String[] args)`。
- l 返回类型：`void`。

#### Tips

在本书中，把类中包含程序入口 `main()` 方法，并且从这个类的 `main()` 方法开始运行的类称为主程序类。例如，在 `dollapp` 应用中，`AppMain` 类就是主程序类。

以下 `main()` 方法都能作为程序入口方法，采取哪种声明方式取决于个人编程的习惯：

```

public static void main(String[] args)
public static void main(String args[])
static public void main(String[] args)

```

`args` 是 `main()` 方法的参数，它是一个 `String` 类型的数组，把这个参数叫作其他的名字也是可以的。关于数组的概念和用法可参见本书的第 14 章（数组）。



此外，由于 `static` 修饰的方法默认都是 `final` 类型（不能被子类覆盖）的，所以在 `main()` 方法前加上 `final` 修饰符也是可以的：

```
final public static void main(String args[])
```

在类中可以通过重载的方式提供多个不作为应用程序入口的 `main()` 方法。关于方法重载的概念参见本书第6章的6.2节（方法重载）。例如在例程2-8的 `AppMain` 类中声明了多个 `main()` 方法。

例程 2-8 AppMain.java

```
package com.abc.dollapp.main;

public class AppMain{
    /** 程序入口 main 方法 */
    public static void main(String args[]){.....}

    /** 非程序入口 main 方法 */
    public static void main(String arg) {.....}
    private int main(int arg) {.....}
}
```

例程2-8的 `AppMain` 类中包含3个 `main()` 方法，第一个方法是作为程序入口的方法，其他两个方法尽管不能作为程序入口，但也是合法的，能通过编译。

### 2.1.6 给 `main()` 方法传递参数

当用 `java` 命令运行 Java 应用程序时，可以在命令行向 `main()` 方法传递参数，格式如下：

```
java classname [args...]
```

以下程序运行 `MainApp` 类，但是没有向 `main()` 方法传递参数：

```
java com.abc.dollapp.main.MainApp
```

此时 `main(String args[])` 方法的参数 `args` 是一个长度为0的数组。如果执行如下命令：

```
java com.abc.dollapp.main.MainApp parameter0 parameter1
```

此时 `main(String args[])` 方法的参数 `args` 是一个长度为2的数组。`args[0]`的值是“parameter0”，`args[1]`的值是“parameter1”。

### 2.1.7 注释语句

在 Java 源文件的任意位置，都可以加入注释语句，Java 编译器会忽略程序中的注释语句。Java 语言提供了以下3种形式的注释：

- ! //text: 从“//”到本行结束的所有字符均作为注释而被编译器忽略。
- ! /\* text \*/: 从“/\*”到“\*/”间的所有字符会被编译器忽略。
- ! /\*\* text \*/: 从“/\*\*”到“\*/”间的所有字符会被编译器忽略。当这类注释出

现在任何声明（如类的声明、类的成员变量的声明或者类的成员方法的声明）之前时，会作为 **JavaDoc** 文档的内容，参见本章 2.3 节（使用和创建 **JavaDoc** 文档）。

### 2.1.8 关键字

Java 语言的关键字是程序代码中的特殊字符，例如在 2.1.3 节的例程 2-7 的 `AppMain.java` 中，`package`、`import`、`public`、`class`、`static` 和 `void` 都是关键字。Java 语言的关键字包括：

- | 用于类和接口的声明：`class`、`extends`、`implements`、`interface`、`enum`。
- | 包引入和包声明：`import`、`package`。
- | 数据类型：`boolean`、`byte`、`char`、`double`、`float`、`int`、`long`、`short`。
- | 某些数据类型的可选值：`false`、`true`、`null`。
- | 流程控制：`break`、`case`、`continue`、`default`、`do`、`else`、`for`、`if`、`return`、`switch`、`while`。
- | 异常处理：`catch`、`finally`、`throw`、`throws`、`try`、`assert`。
- | 修饰符：`abstract`、`final`、`native`、`private`、`protected`、`public`、`static`、`synchronized`、`transient`、`volatile`。
- | 操作符：`instanceof`。
- | 创建对象：`new`。
- | 引用：`this`、`super`。
- | 方法返回类型：`void`。

以上每个关键字都有特殊的作用，例如 `package` 关键字用于包的声明，`import` 关键字用于引入包，`class` 关键字用于类的声明，`void` 关键字表示方法没有返回值。在本书的后面章节还会陆续介绍其他关键字的作用。

Java 语言的保留字是指预留的关键字，虽然它们现在没有作为关键字，但在以后的升级版本中有可能作为关键字。Java 语言的保留字包括 `const` 和 `goto`。

使用 Java 语言的关键字时，有以下值得注意的地方：

- | 所有的关键字都是小写。
- | `friendly`、`sizeof` 不是 Java 语言的关键字，这是有别于 C++ 语言的地方。
- | 程序中的标识符不能以关键字命名。关于标识符的概念参见本章 2.1.9 节（标识符）。

### 2.1.9 标识符

标识符是指程序中包、类、接口、变量或方法的名字。Java 语言要求标识符必须符合以下命名规则：

- | 标识符的首字符必须是字母、下画线（`_`）、符号 `$` 或者符号 `¥`。
- | 标识符由数字（0~9）、从 A~Z 的大写字母、a~z 的小写字母、下画线（`_`），以及美元符 `$` 等组成。

- ❑ 不能把关键字和保留字作为标识符。
- ❑ 标识符没有长度的限制。
- ❑ 标识符是大小写敏感的，这意味着，hello、Hello 和 HELLO 是 3 个不同的标识符。

表 2-1 是一个正误对照表，列举了一些合法标识符和非法标识符。如果程序代码中包含非法标识符，会导致编译错误。

表 2-1 标识符正误对照表

合法标识符	非法标识符	说 明
Try	Try#	标识符中不能包含“#”
GROUP_7	7GROUP	标识符不能以数字符号开头
openDoor	open-door	标识符中不能包含“-”
boolean1	boolean	boolean 是关键字，不能用关键字做标识符

### 2.1.10 编程规范

在 Oracle 的技术网站上公布了 Java 编程规范，网址如下：

<http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>。

编程规范的主要内容如下：

- ❑ 类名和接口名：首字母大写。如果类名由几个单词构成，那么每个单词的首字母大写，其余字母小写，例如：SmartDoll。
- ❑ 方法名和变量名：首字母小写。如果方法名或变量名由几个单词构成，那么除了第一个单词外，其余每个单词的首字母大写，其余字母小写，例如：colorOfDoll。如果变量名指代的实体的数量大于 1，那么采用复数形式，例如：bothEyesOfDoll、allChildren。
- ❑ 包名：采用小写形式，例如：com.abc.dollapp。
- ❑ 常量名：采用大写形式，如果常量名由几个单词构成，单词之间以下画线“\_”隔开，利用下画线可以清晰地分开每个大写的单词。例如：

```
final String DEFAULT_COLOR_OF_DOLL = "yellow";
```

#### Tips

标识符的命名规则是必须遵守的，否则会导致编译错误。而编程规范是推荐遵守的编程习俗，即使不遵守以上编程规范，也不会导致编译错误。

## 2.2 用 JDK 管理 Java 应用

管理 Java 应用是指创建 Java 应用的目录结构、编译、运行，以及发布 Java 应用的操作。表 2-2 显示了 Java 应用的一种常用开发目录结构。

表 2-2 Java 应用的常用目录结构

目录	描述
src 子目录	存放 Java 源文件
classes 子目录	存放编译生成的 Java 类文件
lib 子目录	存放第三方 Java 软件的 JAR 文件
doc 子目录	存放各种帮助文档
doc\api 子目录	存放 JavaDoc 文档
deploy 子目录	存放 Java 应用的打包文件：JAR 文件

对于本章的 dollapp 应用，假定其根目录为\dollapp，在 src 子目录下存放 Java 源文件。值得注意的是。Java 源文件的存放路径必须和包名匹配。如果 Doll 类和 AppMain 类都位于默认包中，那么 Doll.java（参见 2.1 节的例程 2-1）和 AppMain.java（参见 2.1 节的例程 2-2）直接放在 src 根目录下。而对于 com.abc.dollapp.doll.Doll 类（参见 2.1.3 节的例程 2-5），它的源文件 Doll.java 文件应该位于 dollapp\src\com\abc\dollapp\doll 目录下，对于 com.abc.dollapp.main.AppMain 类（参见 2.1.3 节的例程 2-6），它的源文件应该位于 dollapp\src\com\abc\dollapp\main 目录下。图 2-2 为 dollapp 应用的目录结构，其中 classes 目录下的 Doll.class 和 AppMain.class 文件是用 javac 命令编译生成的，deploy 目录下的 dollapp.jar 文件是用 jar 命令打包生成的。

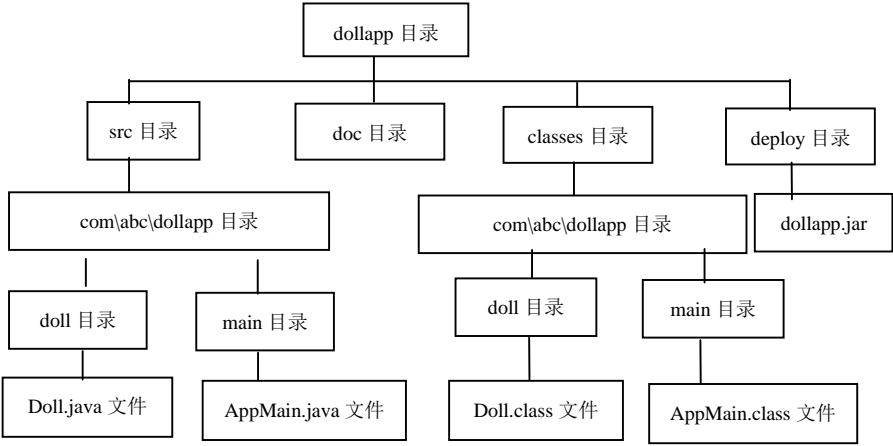


图 2-2 dollapp 应用的目录结构

本节介绍用 JDK 来编译和运行 Java 程序，以及生成 JavaDoc 文档的方法。在实际开发中，一般会使用诸如 Eclipse 等建立在 JDK 基础上的图形化的开发工具软件。本书之所以直接选用 JDK 来编译和运行 Java 程序，一方面是因为它比较便捷，另一方面是因为借助它可以清晰地展示 JDK 编译和运行 Java 程序的基本原理。

2.2.1 JDK 简介以及安装方法

JDK 是 Java Development Kit（Java 开发工具包）的缩写。它为 Java 应用程序提供了基本的开发和运行环境，JDK 也称为 Java 标准开发环境（Java Standard Edition，

JavaSE)。

目前 JDK 的最成熟的版本为 JDK8。如图 2-3 显示了 JDK8 的结构。

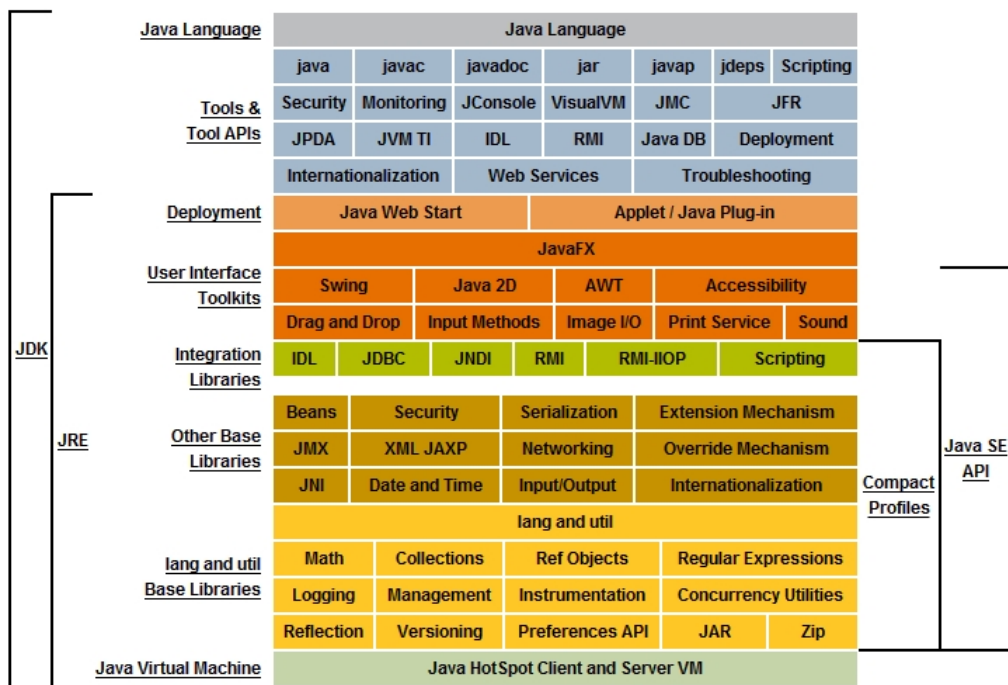


图 2-3 JDK8 的结构

从图 2-3 可以看出，JDK 主要包括以下内容：

- ❶ Java 虚拟机 (Java Virtual Machine)：负责解析和执行 Java 程序。Java 虚拟机可以运行在各种操作系统平台上。
- ❷ JDK 类库 (JavaSE API)：提供了基础的 Java 类及各种实用类。java.lang、java.io、java.util、java.awt、javax.swing 和 java.sql 包中的类都位于 JDK 类库中。
- ❸ 开发工具：这些开发工具都是可执行程序，主要包括：javac.exe（编译工具）、java.exe（运行工具）、javadoc.exe（生成 JavaDoc 文档的工具）和 jar.exe（打包工具）等。

JDK 的官方下载地址为：<http://www.oracle.com/technetwork/java/javase/downloads/index.html>。

为了便于读者下载到与本书配套的 JDK8 软件，在本书的技术支持网站 JavaThinker.net 上也提供了该软件的下载：<http://www.javathinker.net/software/jdk8.exe>。

### Tips

在 JavaSE API 的官方文档中，把 JDK8 也称作 JDK1.8，JDK5 也称作 JDK1.5，以此类推。

假定 JDK 安装到本地后的根目录为 <JAVA\_HOME>，在 <JAVA\_HOME>\bin 目录下提供了以下工具：

- | javac.exe: Java 编译器，把 Java 源文件编译成 Java 类文件。
- | jar.exe: Java 应用的打包工具。
- | java.exe: 运行 Java 程序。
- | javadoc.exe: JavaDoc 文档生成器。

在以下网址对这些工具的用法做了详细介绍：<http://docs.oracle.com/javase/8/docs/technotes/tools/windows/index.html>。

为了便于在 DOS 命令行下直接运行这些工具，可以把<JAVA\_HOME>\bin 目录添加到操作系统的系统环境变量 Path 变量中。假定<JAVA\_HOME>代表“C:\jdk8”目录，在 Windows 操作系统中选择【控制面板】→【系统】→【高级】→【环境变量】→【系统变量】命令，然后编辑其中的 Path 系统变量即可，如图 2-4 所示。

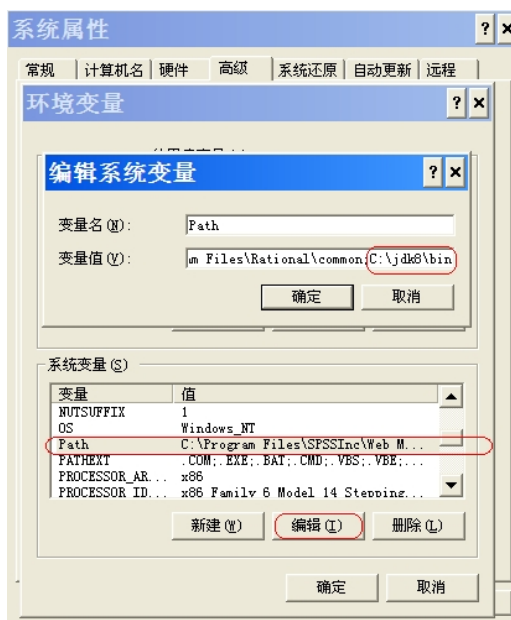


图 2-4 在 Path 系统变量中加入 JDK 的 bin 路径

此外，也可以在 DOS 命令行设置当前的 Path 环境变量。在 Windows 操作系统中选择【开始】→【运行】命令，然后输入“cmd”命令，就会打开一个 DOS 控制台，然后输入以下命令即可：

```
set path=C:\jdk8\bin;%path%
```

## 2.2.2 编译 Java 源文件

javac 命令用于编译 Java 源文件，javac 命令的使用语法如下：

```
javac [ options ] [ sourcefiles ]
```

javac 命令后面跟多个命令选项，以便控制 javac 命令的编译方式。表 2-3 列出了主要命令选项的用法。

表 2-3 javac 命令的选项

命令选项	说明
-nowarn	不输出警告信息。非默认选项。警告信息是编译器针对程序中能编译通过但存在潜在错误的部分提出的信息
-verbose	输出编译器运行中的详细工作信息。非默认选项
-deprecation	输出源程序中使用了不鼓励使用(Deprecated)的 API 的具体位置。非默认选项
-classpath <路径>	覆盖 classpath 环境变量，重新设定用户的 classpath。如果既没有设定 classpath 环境变量，也没有设定 -classpath 选项，那么用户的 classpath 为当前路径
-sourcepath <路径>	指定 Java 源文件的路径
-d <目录>	指定编译生成的类文件的存放目录。值得注意的是，javac 命令并不会自动创建 -d 选项指定的目录，因此必须确保该目录已经存在。如果没有设定此项，编译生成的类文件存放在 Java 源文件所在的目录下
-help	显示各个命令选项的用法

从表 2-3 可以看出，javac 命令的选项有两种形式：一种没有参数，如-nowarn、-verbose 和-deprecation；一种带有参数，如-classpath、-sourcepath 和-d 选项。  
在 DOS 命令行下，运行以下命令，就会编译 AppMain.java：

```
C:\dollapp> javac -sourcepath C:\dollapp\src
               -classpath C:\dollapp\classes
               -d C:\dollapp\classes
               C:\dollapp\src\com\abc\dollapp\main\AppMain.java
```

在以上命令中，-sourcepath 选项指定 Java 源文件的根目录，-classpath 选项指定用户的 classpath，-d 选项指定编译生成的 Java 类文件的存放路径。执行以上命令后，将在 C:\dollapp\classes\com\abc\dollapp\main 目录下生成 AppMain.class 文件。

如果在 javac 命令中再加上-verbose 选项，javac 命令在编译时会输出详细的工作信息：

```
C:\dollapp> javac -verbose
               -sourcepath C:\dollapp\src
               -classpath C:\dollapp\classes
               -d C:\dollapp\classes
               C:\dollapp\src\com\abc\dollapp\main\AppMain.java
```

在 AppMain 类中引入了 com.abc.dollapp.doll.Doll 类，因此 Java 编译器在编译 AppMain 类时，必须先获得 Doll.class。Java 编译器的处理流程如下：



- (1) 由于在 AppMain 类的 import 语句中声明 Doll 类位于 com.abc.dollapp.doll 包中，因此 Java 编译器先到 classpath 根路径下的 com\abc\dollapp\doll 目录下寻找 Doll.class 文件，然后到-sourcepath 选项指定的 src 根目录下的 com\abc\dollapp\doll 目录下寻找 Doll.java 文件。
- (2) 如果同时找到了 Doll.class 文件和 Doll.java 文件，Java 编译器根据 Doll.java

文件的更新日期来判断 Doll.class 有没有过期，如果过期，就重新编译 Doll.java，否则就直接使用 Doll.class。

（3）如果只找到了 Doll.class 文件，Java 编译器就直接使用这个 Doll.class。如果只找到了 Doll.java 文件，Java 编译器就编译这个 Doll.java。

（4）如果既没有找到 Doll.java 文件，也没有找到 Doll.class 文件，就会抛出编译错误，提示无法解析 AppMain 类中的“Doll”符号。

在 javac 命令中可以指定编译多个 Java 源文件，这些文件之间以空格隔开，例如：

```
C:\dollapp> javac -sourcepath C:\dollapp\src
                  -classpath C:\dollapp\classes
                  -d C:\dollapp\classes
C:\dollapp\src\com\abc\dollapp\main\*.java
C:\dollapp\src\com\abc\dollapp\doll\Doll.java
```

以上命令指定编译 C:\dollapp\src\com\abc\dollapp\main 目录下的所有 Java 文件，以及 C:\dollapp\src\com\abc\dollapp\doll 目录下的 Doll.java 文件。

Tips

在本书提供的配套源代码中，在每一章中都有一个包含了 Java 编译命令的 build.bat 批处理文件。直接运行这个文件，就能编译这一章的 Java 源代码。

2.2.3 运行 Java 程序

java 命令用于运行 Java 程序，它会启动 Java 虚拟机，Java 虚拟机加载相关的类，然后调用主程序类的 main()方法。表 2-4 列出了 java 命令的主要命令选项的用法。

表 2-4 java 命令的用法

命令选项	说 明
-classpath <路径>	覆盖 classpath 环境变量，重新设定用户的 classpath。如果既没有设定 classpath 环境变量，也没有设定-classpath 选项，那么用户的 classpath 为当前路径
-verbose	输出运行中的详细工作信息。非默认选项
-D<属性名=属性值>	设置系统属性，例如： <div>java -Duser="Tom" SampleClass</div> 其中“user”为属性名，“Tom”为属性值，SampleClass 为类名。在 SampleClass 中调用 System.getProperty("user")方法就会返回“Tom”属性值
-jar	指定运行某个 JAR 文件中的特定 Java 类
-help	显示各个命令选项的用法

1. 设置 classpath

在运行 Java 程序时，很重要的一个环节是设置 classpath，classpath 代表 Java 类的根路径。java 命令会从 classpath 中寻找所需 Java 类。此外，Java 编译器编译 Java 类时，也会从 classpath 中寻找所需的 Java 类。classpath 的默认值为当前路径，此外，有 3 种显式设置 classpath 的方式：



(1) 在操作系统中定义系统环境变量 classpath。例如在 Windows 中选择【控制面板】→【系统】→【高级系统设置】→【高级】→【环境变量】→【新建】命令，就可以创建系统环境变量 classpath，如图 2-5 所示。



图 2-5 设置系统环境变量 classpath

(2) 在一个 DOS 命令窗口中定义当前环境变量 classpath，例如：

```
C:\> set classpath=C:\classes2
```

(3) 在 java 命令或 javac 命令中通过 -classpath 选项来设置 classpath，例如：

```
C:\> java -classpath C:\classes3; C:\lib\mytools.jar SomeClass
```

Java 虚拟机或 Java 编译器确定 classpath 的流程如下：



- (1) 如果在 java 命令或 javac 命令中设置了 -classpath 选项，就使用这个 classpath。
- (2) 如果在当前 DOS 命令窗口中定义了当前环境变量 classpath，就使用这个 classpath。
- (3) 如果在操作系统中定义了系统环境变量 classpath，就使用这个 classpath。
- (4) 就把当前路径作为 classpath。

由此可见，当前环境变量 classpath 会覆盖系统环境变量 classpath，java 命令或 javac 命令中的 -classpath 选项会覆盖环境变量 classpath，如果希望把当前目录、系统环境变量 classpath，以及当前环境变量 classpath 都添加到 classpath 中，可采用如下方式：

```
C:\> set classpath=%classpath%;C:\classes2
C:\> java -classpath .; %classpath%; C:\classes3; C:\lib\mytools.jar SomeClass
```

第一行命令在设置当前环境变量 classpath 时，先添加了系统环境变量 classpath。在第二行命令中，-classpath 选项中包含多个路径，路径之间以分号隔开，其中第一个

“.” 符号代表当前路径，第二个路径%classpath%代表环境变量 classpath，第三个路径“C:\classes3”为 Java 类文件所在的路径，第四个路径为某个 JAR 文件所在的路径。

在设定 classpath 时，对于 Java 类，只需指定它的根目录，例如，对于 C:\dollapp\classes\com\abc\dollapp\doll\Doll.class，它的 classpath 为 C:\dollapp\classes；对于 JAR 文件，必须指定它的完整路径，例如 mytools.jar 文件的路径为 C:\lib\mytools.jar。

### Tips

JDK 提供了灵活地设置 classpath 的方式，系统环境变量 classpath 是全局性的 classpath，DOS 窗口中定义的环境变量 classpath 只在当前 DOS 窗口中有效，在 javac 或 java 命令中用 -classpath 选项设置的 classpath 则只对当前 javac 或 java 命令有效。

## 2. 运行 AppMain 类

在 DOS 命令行下，输入以下命令，就会执行 AppMain 类的 main()方法：

```
C:\dollapp> java -classpath C:\dollapp\classes com.abc.dollapp.main.AppMain
```

以上命令指定 classpath 为 C:\dollapp\classes。以上命令的运行结果如图 2-6 所示。

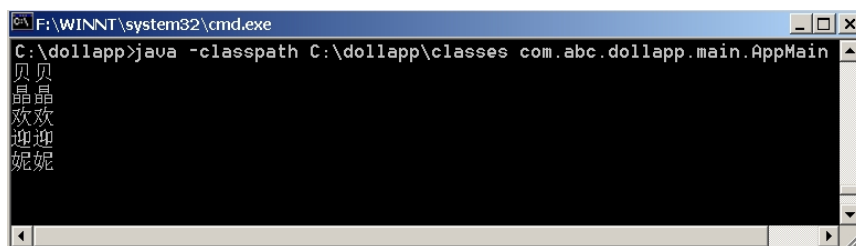


图 2-6 运行 AppMain 类

使用 java 命令时，有以下值得注意的地方：

(1) 必须指定主程序类的完整的名字，比如 com.abc.dollapp.main.AppMain，这样，Java 虚拟机会到 C:\dollapp\classes\com\abc\dollapp\main 目录下寻找 AppMain.class 文件。如果运行以下命令：

```
C:\dollapp> java -classpath C:\dollapp\classes AppMain
```

Java 虚拟机会认为需要运行的 AppMain 类位于默认包中，因此直接在 classpath 的根路径下寻找 AppMain.class 文件，如果找不到，就会抛出错误。

(2) 在 classpath 中，类文件的存放位置必须和包名匹配，不能随意改变它们的存放位置。如果把 com.abc.dollapp.main.AppMain 类的 AppMain.class 类文件移动到 C:\dollapp\classes 根目录下，然后运行命令：

```
C:\dollapp> java -classpath C:\dollapp\classes AppMain
```

Java 虚拟机会认为需要运行的 AppMain 类位于默认包中，因此直接在 classpath 的根路径下寻找 AppMain.class 文件，尽管找到了这个文件，但是在解析和验证这个文件时，发现其中的 package 语句声明 AppMain 类位于 com.abc.dollapp.main 包中，而不在

默认包中，因此还是会抛出错误。

(3) 在 java 命令中指定的 Java 类必须具有作为程序入口的 main() 方法。

## 2.2.4 给 Java 应用打包

JDK 的 jar 命令能够把 Java 应用打包成一个文件，这个文件的扩展名为 .jar。这种打包文件被称为 JAR (Java Archive) 文件，它独立于任何操作系统平台，而且支持压缩格式。给 Java 应用打包的好处在于：便于发布 Java 应用，提高在网络上传输 Java 应用的速度。在 DOS 命令行转到 C:\dollapp\classes 目录下，然后运行如下命令：

```
C:\dollapp\classes> jar -cvf C:\dollapp\deploy\dollapp.jar *
```

以上 jar 命令会把 C:\dollapp\classes 目录下（包括其子目录下）的所有类文件打包为 dollapp.jar 文件，把它存放在 C:\dollapp\deploy 目录下。

java 命令和 javac 命令会读取 JAR 文件中的 Java 类。例如以下命令把 dollapp.jar 添加到 classpath 中，然后运行其中的 AppMain 类：

```
C:\dollapp> java -classpath C:\dollapp\deploy\dollapp.jar  
com.abc.dollapp.main.AppMain
```

jar 命令还具有展开 JAR 文件的功能，如果运行如下命令：

```
C:\dollapp\classes> jar -xvf C:\dollapp\deploy\dollapp.jar
```

以上命令会重新展开 dollapp.jar 文件中的内容，在展开内容中有一个 MANIFEST.MF 文件，它包含了描述 JAR 文件的信息。

本章 2.2.3 节的表 2-4 提到 java 命令的 -jar 选项直接指定运行某个 JAR 文件中的特定 Java 类，在这种情况下，在这个 JAR 文件的 MANIFEST.MF 文件中必须包含主程序类的名字，以下是制作 JAR 文件并运行这个 JAR 文件的步骤：



(1) 在 classes 目录下创建一个 Manifest.txt 文件，文件中包含如下内容：

```
Main-Class: com.abc.dollapp.main.AppMain
```

以上内容表明 JAR 文件的主程序类为 com.abc.dollapp.main.AppMain 类。为了使 jar 命令能正确解析 Manifest.txt 文件，以上内容必须以换行结束。

(2) 在 C:\dollapp\classes 目录下，运行如下 jar 命令：

```
C:\dollapp\classes> jar -cvfm C:\dollapp\deploy\dollapp.jar Manifest.txt *
```

以上 jar 命令会把 Manifest.txt 文件中的内容添加到 MANIFEST.MF 文件中，并且在 C:\dollapp\deploy 目录下生成 dollapp.jar 文件。

(3) 在 C:\dollapp 目录下，运行如下命令：

```
C:\dollapp> java -jar C:\dollapp\deploy\dollapp.jar
```

以上 java 命令根据 dollapp.jar 文件中的 MANIFEST.MF 文件的信息，确定主程序

类为 AppMain 类，因此执行这个类的 main() 方法。

## 2.3 使用和创建 JavaDoc 文档

Java 类通过 JavaDoc 文档来对外公布自身的用法，JavaDoc 文档是基于 HTML 格式的帮助文档。例如，图 2-7 为 JDK 的 Java 基本包中的 Object 类的 JavaDoc 文档，这一文档描述了 Object 类，以及它的各个方法的功能、用法及注意事项。JDK8 的 JavaDoc 文档的地址为：<http://docs.oracle.com/javase/8/docs/api/index.html>。

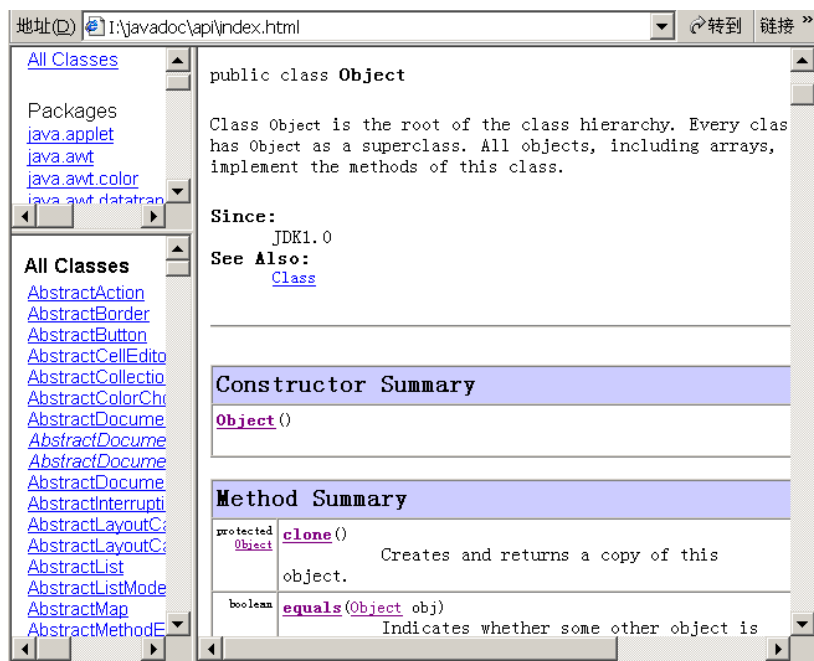


图 2-7 Object 类的 JavaDoc 文档

### Tips

JavaDoc 文档是供 Java 开发人员阅读的，他们通过 JavaDoc 文档来了解其他人员开发的类的用法。Java 开发人员应该养成经常查阅 JavaDoc 文档的良好习惯。

对于用户创建的 Java 类，如何编写这种 HTML 格式的 JavaDoc 文档呢？手工编写 JavaDoc 文档显然是很费力的事。幸运的是，JDK 中提供了一个 javadoc.exe 程序，它能够识别 Java 源文件中符合特定规范的注释语句，根据这些注释语句自动生成 JavaDoc 文档。

在 Java 源文件中，只有满足特定规范的注释，才会构成 JavaDoc 文档。这些规范包括：

(1) 注释以 “/\*” 开始，并以 “\*/” 结束，里面可以包含普通文本、HTML 标记和 JavaDoc 标记。例如以下注释将被 javadoc 命令解析为 JavaDoc 文档：

```

/**
 * <p><strong>SmartDoll</strong>代表智能福娃，它能够发出用户指定的声音。</p>
 * @author 孙卫琴
 * @version 3.0
 * @since 1.0
 * @see com.abc.dollapp.doll.Doll
 */
public class SmartDoll extends Doll{.....}

```

以上注释用于描述 SmartDoll 类的作用，其中<p>和<strong>为 HTML 标记，@author、@version、@since 和@see 为 Javadoc 标记。javadoc 命令能够解析以上注释，最后生成的 Javadoc 文档如图 2-8 所示。



```

java.lang.Object
com.abc.dollapp.doll.Doll
com.abc.dollapp.doll.extend.SmartDoll

public class SmartDoll
extends Doll
SmartDoll 代表智能福娃，它能够发出用户指定的声音。
从以下版本开始:
1.0

版本:
3.0

作者:
孙卫琴

另请参阅:
Doll

```

图 2-8 描述 SmartDoll 类的 Javadoc 文档

(2) javadoc 命令只处理 Java 源文件中在类声明、接口声明、成员方法声明、成员变量声明，以及构造方法声明之前的注释，忽略位于其他地方的注释。例如，在以下程序代码中，只有粗体字部分标识的注释语句会构成 Javadoc 文档。变量 var1 在 method()方法中定义，是局部变量，因此在它之前的注释语句会被 javadoc 命令忽略。

```

/** 类的注释语句 */
public class JavaDocSample{
    /** 成员变量的注释语句 */
    public int var;

    /** 构造方法的注释语句 */
    public JavaDocSample(){ }

    /** 成员方法的注释语句 */
    public void method(){
        /** 局部变量的注释语句 */
        int var1=0;
    }
}

```

### 2.3.1 JavaDoc 标记

在构成 JavaDoc 文档的注释语句中，可以使用 JavaDoc 标记来描述作者、版本、方法参数和方法返回值等信息。表 2-5 列出了常见的 JavaDoc 标记的作用。

表 2-5 JavaDoc 标记

JavaDoc 标记	描述
@version	指定版本信息
@since	指定最早出现在哪个版本
@author	指定作者
@see	生成参考其他 JavaDoc 文档的链接
@link	生成参考其他 JavaDoc 文档的链接，它和@see 标记的区别在于，@link 标记能够嵌入到注释语句中，为注释语句中的特定词汇生成链接
@deprecated	用来标明被注释的类、变量或方法已经不提倡使用，在将来的版本中有可能被废弃
@param	描述方法的参数
@return	描述方法的返回值
@throws	描述方法抛出的异常，指明抛出异常的条件

下面通过一个 SmartDoll 类的例子来介绍常用的 JavaDoc 标记的用法。SmartDoll 继承了 Doll 类，SmartDoll 位于 com.abc.dollapp.doll.extend 包中。假定伴随着 dollapp 应用的升级换代，SmartDoll 类也先后由 1.0 版本、2.0 版本最后升级到 3.0 版本。SmartDoll 类的 JavaDoc 文档是供其他开发人员阅读的，在 JavaDoc 文档中应该包含描述 SmartDoll 类的具体用法的信息，以及版本升级的信息。这样，如果其他开发人员原先使用的是 SmartDoll 类的 1.0 版本，通过阅读 3.0 版的 JavaDoc 文档，就会知道如何对自己的程序做相应的升级，改为使用 SmartDoll 类的 3.0 版本。例程 2-9 是 SmartDoll 类的源程序。

例程 2-9 SmartDoll.java

```

/*
 * 版权 2005-2025 www.javathinker.net
 * 本程序采用 GPL 协议，你可以从以下网址获得该协议的内容：
 * http://www.gnu.org/copyleft/gpl.html
 */
package com.abc.dollapp.doll.extend;
import com.abc.dollapp.doll.Doll;

/**
 * <p><strong>SmartDoll</strong> 代表智能福娃，它能够发出用户指定的声音。</p>
 * @author 孙卫琴
 * @version 3.0
 * @since 1.0
 * @see com.abc.dollapp.doll.Doll
 */
public class SmartDoll extends Doll{

```

```

/**
 * 代表智能福娃默认情况下所说的话
 */
protected String word;

/**
 * 构造一个智能福娃，未设定默认情况下所说的话
 */
public SmartDoll(String name){
    super(name);
};

/**
 * 构造智能福娃的同时，指定默认情况下所说的话
 * @param word 默认情况下所说的话
 */
public SmartDoll(String name,String word){
    super(name);
    this.word=word;
};

/**
 * 获得默认情况下所说的话
 * @return 返回默认情况下所说的话
 * @see #setWord
 * @deprecated 该方法已经被废弃
 */
public String getWord(){
    return this.word;
}

/**
 * 设置默认情况下所说的话
 * @param word 默认情况下所说的话
 * @see #getWord
 * @since 2.0
 */
public void setWord(String word){
    this.word=word;
}

/**
 * <ul>
 * <li>如果{ @link #word word 成员变量}不为 null,
 *     就调用{ @link #speak(String) speak(String)方法}</li>
 * <li>如果{ @link #word word 成员变量}为 null,
 *     就调用{ @link com.abc.dollapp.doll.Doll#speak() super.speak()方法}</li>
 * </ul>
 */
public void speak(){
    if(this.word!=null){
        try{
            speak(word);
        }catch(Exception e){}
    }else

```

```

        super.speak();
    }
    */
    public void speak(){
        if(this.word!=null){
            try{
                speak(word);
            }catch(Exception e){}
        }
        else
            super.speak();
    }

    /**
     * @param word 指定智能福娃该说的话
     * @return 智能福娃已说的话
     * @exception Exception 如果 word 参数为 null，就抛出该异常
     */
    public String speak(String word) throws Exception{
        if(word==null)
            throw new Exception("不知道该说啥");
        System.out.println(word);
        return word;
    }
}

```

在 SmartDoll 的源程序中，位于 package 语句前的注释会被 javadoc 命令忽略，其余的注释均会构成 JavaDoc 文档。下面依次介绍各个 JavaDoc 标记的用法。

(1) @version 指明该程序的版本，@since 指明程序代码最早出现在哪个版本中。例如：

```

/**
 * <p><strong>SmartDoll</strong>代表智能福娃，它能够发出用户指定的声音。</p>
 * @author 孙卫琴
 * @version 3.0
 * @since 1.0
 * @see com.abc.dollapp.doll.Doll
 */
public class SmartDoll extends Doll{...}

```

以上注释表明 SmartDoll 类的最早版本为 1.0，当前版本为 3.0。对于 SmartDoll 类中的成员变量和成员方法，如何标明它们是在哪个版本中出现的呢？一种惯例做法是，默认情况下，假定它们在类的最早版本中就出现了，否则，就用 @since 标记明确地标明它们最早出现的版本。例如在以下注释中没有使用 @since 标记，表明 word 变量在 SmartDoll 类的 1.0 版本中就已经存在了：

```

/**
 * 代表智能福娃默认情况下所说的话
 */
protected String word;

```

再例如以下注释采用 @since 标记显式指明：setWord()方法是在 SmartDoll 类的 2.0 版本中才添加进去的。



```
/**
 * 设置默认情况下所说的话
 * @param word 默认情况下所说的话
 * @return 无返回值
 * @see #getWord
 * @since 2.0
 */
public void setWord(String word){...}
```

(2) @deprecated 标记用来标明被注释的类、变量或方法已经不提倡使用。例如以下注释中的 @deprecated 标记表明 getWord() 方法已经被废弃：

```
/**
 * 获得默认情况下所说的话
 * @return 返回默认情况下所说的话
 * @see #setWord
 * @deprecated 该方法已经被废弃
 */
public String getWord(){
    return this.word;
}
```

以上注释对应的 JavaDoc 文档如图 2-9 所示。

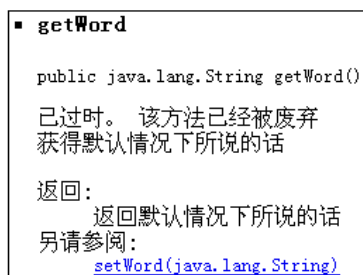


图 2-9 getWord() 方法的 JavaDoc 文档

如果类 A 访问了类 B 中被标记为 Deprecated 的方法，那么在编译类 A 时，Java 编译器会生成一些警告信息，建议开发人员最好不要使用这些被废弃 (Deprecated) 的方法。javac 命令中的 -deprecation 选项用于显示类 A 中访问类 B 的废弃方法的详细位置。

(3) @see 标记用于生成参考其他 JavaDoc 文档的链接，例如 “@see #setWord” 标记将生成参考 setWord() 方法的链接，参见图 2-9 所示。@see 标记有 3 种用法：

① 链接其他类的 JavaDoc 文档，必须给出类的完整类名，例如：

```
@see com.abc.dollapp.doll.Doll
```

② 链接当前类的方法或变量的 JavaDoc 文档，例如：

```
@see #setWord
@see #word
```

③ 链接其他类的方法或变量的 JavaDoc 文档，例如：

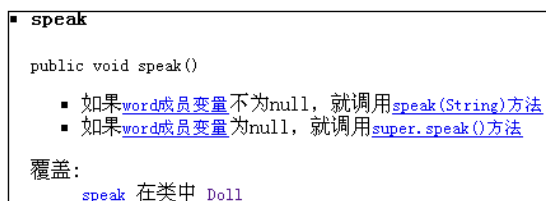
```
@see com.abc.dollapp.doll.Doll#speak
```

(4) @link 标记和@see 标记一样，也能生成参考其他 JavaDoc 文档的链接。两者的区别在于，@link 标记能够嵌入到注释语句中，为注释语句中的特定词汇生成链接。例如：

```
/**
 * <ul>
 * <li>如果{ @link #word word 成员变量}不为 null,
 *     就调用{ @link #speak(String) speak(String)方法}</li>
 * <li>如果{ @link #word word 成员变量}为 null,
 *     就调用{ @link com.abc.dollapp.doll.Doll#speak() super.speak()方法}</li>
 * </ul>
 */
public void speak(){...}
```

以上注释对应的 JavaDoc 文档如图 2-10 所示。其中第一个@link 标记所在的注释行对应的 JavaDoc 文档的源代码为：

```
<li>
  如果
  <A HREF=".../com/abc/dollapp/doll/extend/SmartDoll.html#word">
    <CODE>word 成员变量</CODE>
  </A>
  不为 null，就调用
  <A HREF=".../com/abc/dollapp/doll/extend/SmartDoll.html#speak(java.lang.String)">
    <CODE>speak(String)方法</CODE>
  </A>
</li>
```



```
▪ speak
public void speak()
▪ 如果word成员变量不为null, 就调用speak(String)方法
▪ 如果word成员变量为null, 就调用super.speak()方法
覆盖:
speak 在类中 Doll
```

图 2-10 speak()方法的 JavaDoc 文档

## Tips

图 2-10 中的覆盖项是 javadoc 命令自动生成的，而非由特定的 JavaDoc 标记指定。它表明 SmartDoll 类的 speak()方法覆盖了 Doll 类的 speak()方法。

(5) @param、@return 和@throws 标记分别用来描述方法的参数、返回值及抛出异常的条件。例如：

```
/**
 * @param word 指定智能福娃该说的话
 * @return 智能福娃已说的话
 * @exception Exception 如果 word 参数为 null，就抛出该异常
 */
public String speak(String word) throws Exception{...}
```

以上注释对应的 JavaDoc 文档如图 2-11 所示。

▪ **speak**

```
public java.lang.String speak(java.lang.String word)
    throws java.lang.Exception
```

参数:  
word - 指定智能福娃该说的话

返回:  
智能富娃已说的话

抛出:  
java.lang.Exception - 如果word参数为null, 就抛出该异常

图 2-11 speak(String word)方法的 JavaDoc 文档

2.3.2 javadoc 命令的用法

javadoc 命令和 javac 命令一样，也包含了许多命令选项，表 2-6 列举了常用命令选项的用法。

表 2-6 javadoc 命令的选项

命令选项	作用	是否为默认选项/选项参数的默认值
-public	仅为 public 访问级别的类及类的成员生成 JavaDoc 文档	非默认选项
-protected	仅为 public 和 protected 访问级别的类及类的成员生成 JavaDoc 文档	默认选项
-package	仅为 public、protected，以及默认访问级别的类及类的成员生成 JavaDoc 文档	非默认选项
-private	为 public、protected、默认，以及 private 访问级别的类和类的成员生成 JavaDoc 文档	非默认选项
-version	解析@version 标记	非默认选项
-author	解析@author 标记	非默认选项
-splitindex	把索引文件划分为每个字母对应一个索引文件	非默认选项
-sourcepath <pathlist>	指定 Java 源文件的路径	参数的默认值为当前目录
-classpath <pathlist>	指定 classpath	参数的默认值为当前目录
-d <directory>	指定 JavaDoc 文档的输出目录	参数的默认值为当前目录

Tips

确切地说，顶层类（相对于内部类，参见第 12 章（内部类））的访问级别只能是 public 和默认级别，而类的成员（包括构造方法、成员变量和成员方法）的访问级别可以是 public、protected、默认和 private 级别。表 2-6 中对 -public、-protected、-package 和 -private 等选项做了粗略的描述。

javadoc 命令的使用格式如下：

```
javadoc [options] [packagenames] [sourcefiles]
```

javadoc 命令既可以处理包，也可以处理 Java 源文件。例如在本章的例子中，共有

3 个类：

- ┆ com.abc.dollapp.doll.Doll 类。
- ┆ com.abc.dollapp.doll.extend.SmartDoll 类。
- ┆ com.abc.dollapp.main.AppMain 类。

以下 javadoc 命令设定了 3 个包：

```
C:\dollapp> javadoc -author -version
               -sourcepath src
               -doc \api
               com.abc.dollapp.doll
               com.abc.dollapp.doll.extend
               com.abc.dollapp.main
```

javadoc 命令会依次处理每个包的所有 Java 类，值得注意的是，com.abc.dollapp.doll 和 com.abc.dollapp.doll.extend 是不同的包，当 javadoc 命令处理 com.abc.dollapp.doll 包中的类时，不会自动处理它的 com.abc.dollapp.doll.extend 子级包中的类，因此必须在命令行再单独指定它。以上 javadoc 命令会在 doc\api 目录下生成 JavaDoc 文档，它的首页为 index.html 文件，如图 2-12 所示。

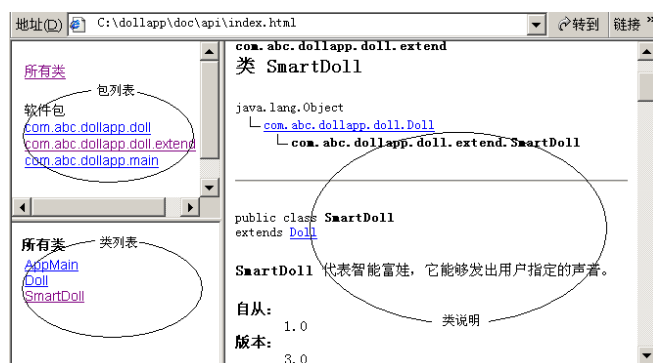


图 2-12 dollapp 应用的 JavaDoc 文档

图 2-12 的 JavaDoc 文档分成 3 部分：包列表、类列表和类说明。在包列表中选择了某个包之后，类列表中就会列出该包中的所有类；在类列表中选择了某个类之后，类说明部分就会显示出该类的详细文档。

以下 javadoc 命令设定了 3 个 Java 类：

```
C:\dollapp> javadoc -author -version
               -sourcepath src
               -d doc\api
               src\com\abc\dollapp\doll\Doll.java
               src\com\abc\dollapp\doll\extend\SmartDoll.java
               src\com\abc\dollapp\main\AppMain.java
```

以上 javadoc 命令只会为指定的 3 个 Java 类生成 JavaDoc 文档，如果指定的 Java 类所在包中还包含其他类，javadoc 命令不会自动处理它们。如果要为 doll 子目录下的所有 Java 类生成 JavaDoc 文档，那么可以指定路径 “src\com\abc\dollapp\doll\\*.java”。

下面再介绍 javadoc 命令的几个选项的用法。

(1) `-public`、`-protected`、`-package` 和 `-private` 这4个选项用于指定输出哪些访问级别的类和成员的 `JavaDoc` 文档，其中 `-protected` 选项为默认选项，也就是说，默认情况下，`javadoc` 命令只会输出访问级别为 `public` 和 `protected` 的类和成员的 `JavaDoc` 文档，如果希望输出所有访问级别的类和成员的 `JavaDoc` 文档，必须显式设定 `-private` 选项：

```
C:\dollapp>javadoc -private -version -author ...
```

(2) `-version` 和 `-author` 选项指定在 `JavaDoc` 文档中包含由 `@version` 和 `@author` 标记指示的内容。这两个选项不是默认选项，也就是说，默认情况下，`javadoc` 命令会忽略注释中的 `@version` 和 `@author` 标记，因此生成的 `JavaDoc` 文档中不包含版本和作者信息。

(3) `-splitindex` 选项将索引分为每个字母对应一个索引文件。这个选项不是默认选项，也就是说，默认情况下，索引文件只有一个，该文件中包含所有索引内容。例如在 `dollapp` 应用的 `doc\api` 目录下有一个 `index-all.html` 文件，它就是索引文件，如图 2-13 所示。当 `JavaDoc` 文档内容不多的时候，不一定要使用 `-splitindex` 选项。但是，如果文档内容非常多，这个 `index-all.html` 索引文件将变得非常庞大，此时使用 `-splitindex` 选项，会使得 `javadoc` 命令按照字母来为索引内容分类，每个字母对应一个索引文件，这些索引文件放在 `doc\api\index-files` 目录下。



图 2-13 index-all.html 索引文件

## 2.4 Java 虚拟机运行 Java 程序的基本原理

Java 虚拟机 (Java Virtual Machine, JVM) 是由 JDK 提供的一个软件程序。虚拟机的任务是执行 Java 程序，如图 2-14 所示。

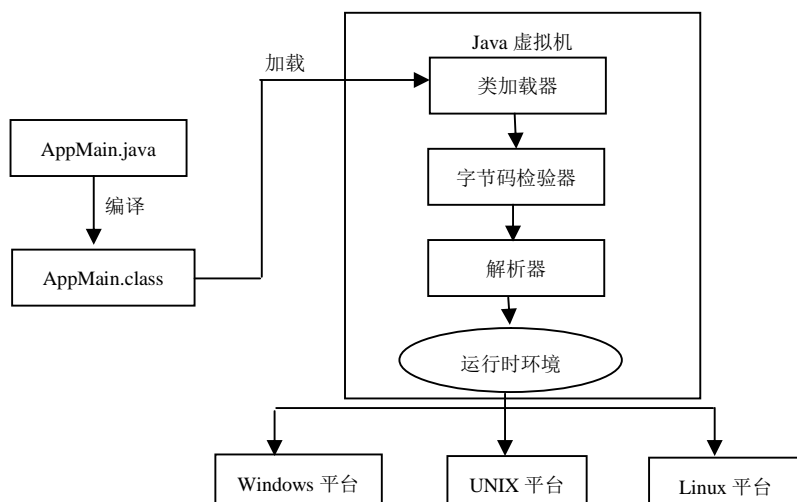


图 2-14 Java 虚拟机执行 Java 程序的过程

从图 2-14 可以看出，由 Java 源文件编译出来的类文件可以在任意一种平台上运行，Java 语言之所以有这种跨平台的特点，要归功于 Java 虚拟机。Java 虚拟机封装了底层操作系统的差异，不管是在哪种平台上，都按以下同样的步骤来运行程序：

### 步骤

- (1) 把.class 文件中的二进制数据加载到内存中。
- (2) 对类的二进制数据进行验证。
- (3) 解析并执行指令。

Java 虚拟机提供了程序的运行时环境，运行时环境中最重要的一个资源是运行时数据区。运行时数据区是操作系统为 Java 虚拟机进程分配的内存区域，Java 虚拟机管辖着这块区域，它把该区域又进一步划分为多个子区域，主要包括堆区、方法区和 Java 栈区等。在堆区中存放对象，方法区中存放类的类型信息，类型信息包括静态变量和方法信息等，方法信息中包含类的所有方法的字节码。

当运行“java AppMain”命令时，就启动了一个 Java 虚拟机进程，该进程首先从 classpath 中找到 AppMain.class 文件，读取这个文件中的二进制数据，把 AppMain 类的类型信息存放到运行时数据区的方法区中。这一过程称为 AppMain 类的加载过程。

Java 虚拟机加载了 AppMain 类后，还会对 AppMain 类进行验证及初始化，第 10 章的 10.2 节（类的加载、连接和初始化）详细介绍了这一过程。Java 虚拟机接着定位到方法区中 AppMain 类的 main()方法的字节码，执行它的指令。main()方法的第一条语句为：

```
Doll beibei=new Doll("贝贝");
```

以上语句创建一个 Doll 实例，并且使引用变量 beibei 引用这个实例。Java 虚拟机执行这条语句的步骤如下：

## 步骤

(1) 搜索方法区，查找 Doll 类的类型信息，由于此时不存在该信息，因此 Java 虚拟机先加载 Doll 类，把 Doll 类的类型信息存放在方法区。

(2) 在堆区中为一个新的 Doll 实例分配内存，这个 Doll 实例持有指向方法区的 Doll 类的类型信息的指针。这里的指针实际上指的是 Doll 类的类型信息在方法区中的内存地址，在 Doll 实例的数据区存放了这一地址。

(3) beibei 变量在 main()方法中定义，它是局部变量，它被添加到执行 main()方法的主线程的 Java 方法调用栈中。关于线程的概念参见本书第 13 章（多线程与并发）。这个 beibei 变量引用堆中的 Doll 实例，也就是说，它持有指向 Doll 实例的指针。如图 2-15 显示了此时 Java 虚拟机的运行时数据区的内存分配。

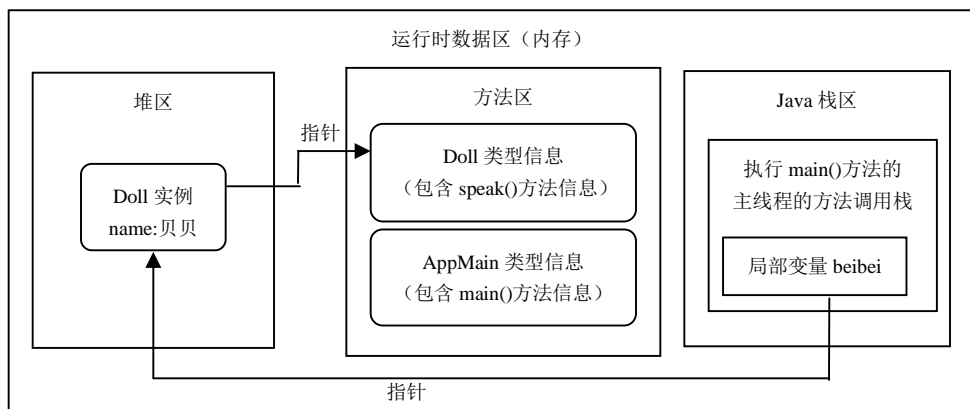


图 2-15 Java 虚拟机的运行时数据区

接下来 Java 虚拟机在堆区创建其他 4 个 Doll 实例，然后依次执行它们的 speak() 方法。当 Java 虚拟机执行 beibei.speak() 方法时，Java 虚拟机根据局部变量 beibei 持有的指针，定位到堆区中的 Doll 实例，再根据 Doll 实例持有的指针，定位到方法区中 Doll 类的类型信息，从而获得 speak() 方法的字节码，接着执行 speak() 方法包含的指令。

## 2.5 小结

本章通过简单的 dollapp 应用例子，介绍了创建、编译、运行和发布 Java 应用的过程，此外还介绍了生成 JavaDoc 文档的步骤。读者应该掌握以下内容：

- 1 Java 源文件的结构。在一个 Java 源文件中可以包含一个 package 语句、多个 import 语句，以及多个类和接口的声明，只能有一个类或接口为 public 类型。
- 1 Java 应用的开发目录结构。目录结构中包含 Java 源文件根目录（src）、Java 类文件根目录（classes）和帮助文档根目录（doc）等。其中 Java 源文件的存放位置应该和它的 package 语句声明的包名一致。
- 1 JDK 中的常用工具的法。javac、java、jar 命令分别用于编译、运行和打包

Java 应用，javadoc 命令能够解析 Java 源文件中的特定注释行，生成 JavaDoc 文档。在使用 javac 和 java 命令时，一个重要的环节是设置 classpath。在 javac 和 java 命令中的 -classpath 选项的优先级别最高，其次是在当前 DOS 窗口中设置的环境变量 classpath，其次是操作系统的系统环境变量 classpath。在设置 Java 类的 classpath 时，只需指定根路径，例如 com.abc.dollapp.doll.Doll 类的类文件的路径为：C:\dollapp\classes\com\abc\dollapp\doll\Doll.class

那么 Doll 类的 classpath 为 C:\dollapp\classes。

在设置 JAR 文件的 classpath 时，需要指定完整路径，例如 dollapp.jar 文件的路径为：C:\dollapp\deploy\dollapp.jar

那么 dollapp.jar 文件的 classpath 为 C:\dollapp\deploy\dollapp.jar。javac 或 java 命令能够读取 dollapp.jar 文件中的所有 Java 类。

## 2.6 思考题

1. 把一个类放在包里有什么作用？
2. JavaDoc 文档是不是为软件的终端用户提供的使用指南？
3. 对于 com.abc.dollapp.AppMain 类，使用以下命令进行编译，

```
java -d C:\classes -sourcepath C:\dollapp\src C:\dollapp\src\com\abc\dollapp\AppMain.java
```

编译出来的.class 文件位于什么目录下？

4. 对于以上编译出来的 AppMain 类，以下哪个 java 命令能正确地运行它？

- a) java C:\classes\com\abc\dollapp\AppMain.class
- b) java -classpath C:\classes AppMain
- c) java -classpath C:\classes\com\abc\dollapp AppMain
- d) java -classpath C:\classes com.abc.dollapp.AppMain

5. 以下哪个 main() 方法的声明能够作为程序的入口方法？

- a) public static void main()
- b) public static void main(String[] string)
- c) public static void main(String args)
- d) static public int main(String[] args)
- e) static void main(String[] args)

6. 假定以下程序代码都分别放在 MyClass.java 文件中，哪些程序代码能够编译通过？

- a)

```
import java.awt.*;
package myPackage;
class MyClass { }
```

- b)

```
package myPackage;
```



```
import java.awt.*;
class MyClass{ }
```

c)

```
/*This is a comment */
package myPackage;
import java.awt.*;
public class MyClass{ }
```

d)

```
/*This is a comment */
package myPackage;
import java.awt.*;
public class OtherClass{ }
```

7. 对于以下 Myprog 类，运行命令 “java Myprog good morning”，将会出现什么情况？

```
public class Myprog{
    public static void main(String argv[]){
        System.out.println(argv[2]);
    }
}
```

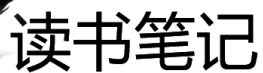
8. 下面哪些是 Java 的关键字？

a) default   b) NULL   c) String   d) throws   e) long   f) true

9. 当 AppMain 类的 main() 方法创建了 5 个 Doll 对象时，运行时数据区的数据是如何分布的？参照 2.4 节的图 2-15，画出此时运行时数据据区的状态图。

10. 下面哪些是合法的 Java 标识符？

a) #\_pound   b) \_underscore   c) 5Interstate   d) Interstate5   e) \_5\_   f) class

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.