

04830180-编译实习

02. Java和MiniJava

黄 骏

jun.huang@pku.edu.cn

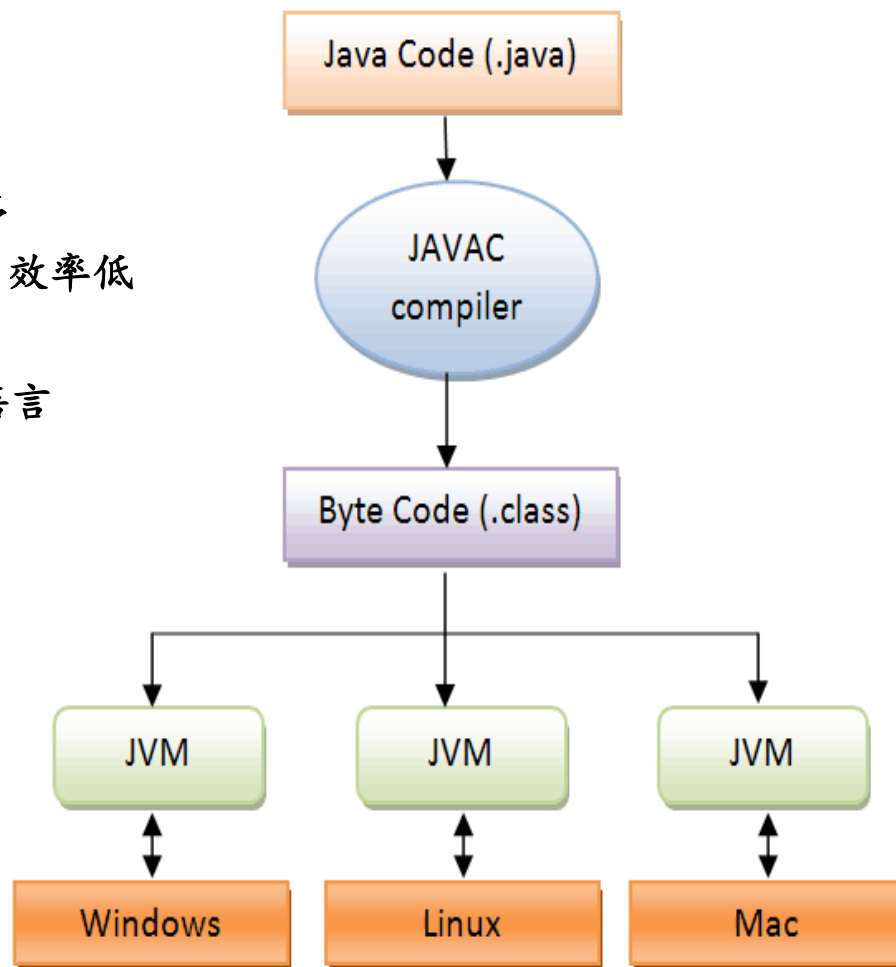
高能效计算与应用中心

北京大学信息科学与技术学院

理科五号楼515S

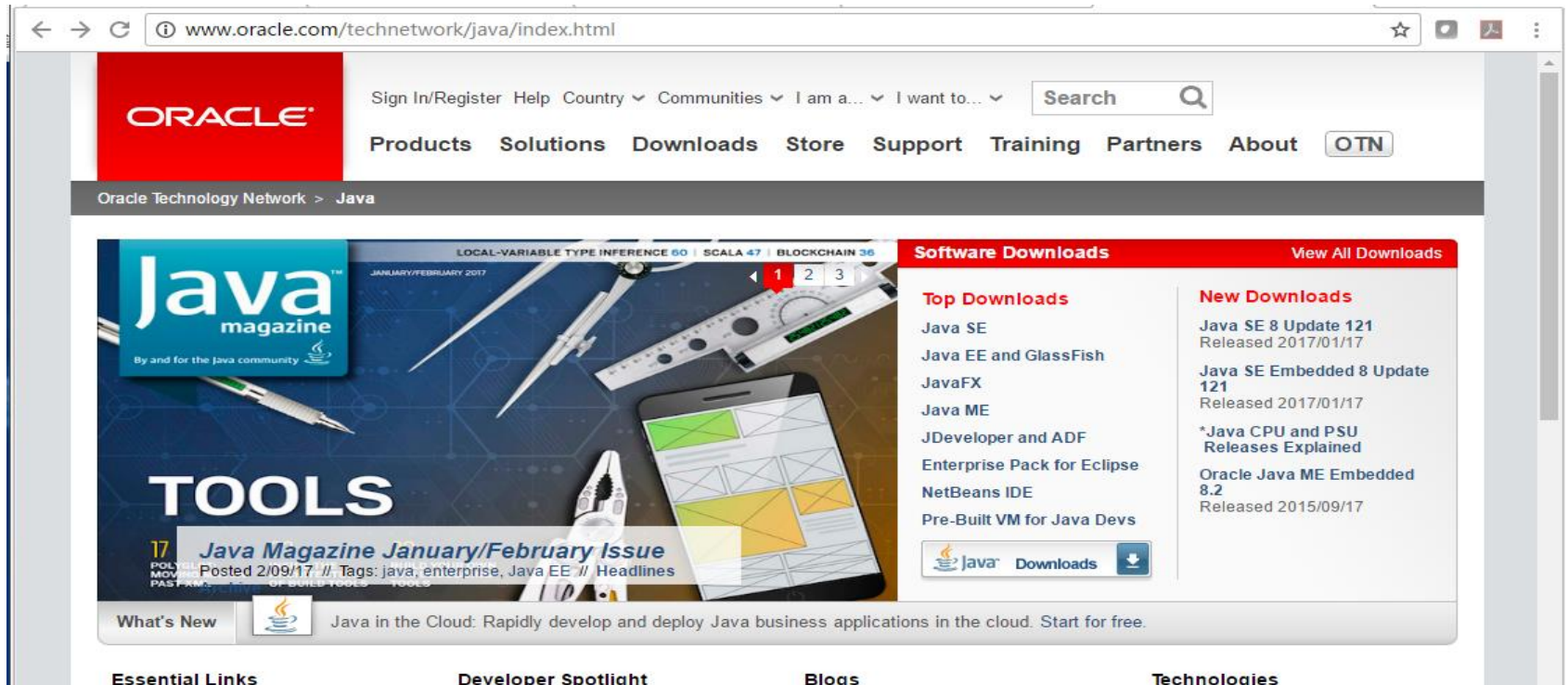
Java程序运行机制

- 计算机的高级编程语言类型：
 - 解释型：
 - 运行时翻译并直接执行源程序
 - Matlab, Python, 平台兼容性, 效率低
 - 编译型：
 - 执行前将源程序翻译为机器语言
 - C/C++, 高效, 兼容性差
- Java 语言是两种类型的结合;
- 编译和执行过程
 - Java源程序
 - Byte Code
 - 机器代码



Java Development Kit (JDK)

- Including JRE
- Can be downloaded from Oracle website




The screenshot shows the Oracle Technology Network (OTN) Java page. The browser address bar displays www.oracle.com/technetwork/java/index.html. The Oracle logo is in the top left. Navigation links include Sign In/Register, Help, Country, Communities, I am a..., I want to..., Search, Products, Solutions, Downloads, Store, Support, Training, Partners, About, and OTN. The main content area features a large banner for Java Magazine with the word "TOOLS" and a link to the January/February issue. To the right, the "Software Downloads" section lists "Top Downloads" and "New Downloads".

Software Downloads [View All Downloads](#)

Top Downloads	New Downloads
Java SE	Java SE 8 Update 121 Released 2017/01/17
Java EE and GlassFish	Java SE Embedded 8 Update 121 Released 2017/01/17
JavaFX	*Java CPU and PSU Releases Explained
Java ME	Oracle Java ME Embedded 8.2 Released 2015/09/17
JDeveloper and ADF	
Enterprise Pack for Eclipse	
NetBeans IDE	
Pre-Built VM for Java Devs	

[Java Downloads](#)

What's New  Java in the Cloud: Rapidly develop and deploy Java business applications in the cloud. Start for free.

Essential Links **Developer Spotlight** **Blogs** **Technologies**

<http://java.sun.com>

程序的编辑、编译与运行

- 程序编辑：编辑器——文件名与**public class**的类名一致。区分大小写
- 设定**path**和**classpath**
 - 前者是命令的路径（**java, javac, ...**）
 - 后者是所要引用的类的路径
 - 可系统配置或编译过程中手动添加（**javac -cp ... HellowWorld.java**）
- 程序编译——转换为字节码文件，扩展名**.class**。其中包含**java**虚拟机的指令。编译可以使用**JDK**工具**javac.exe**。
- 程序的运行——执行 **.class**文件中的指令的过程。

Java

- 类和实例
- 继承、隐藏和覆盖
- 多态
- 接口
- 包

类(Class)

- 类(class)是具有相同属性和行为的对象的抽象
- 成员：字段和方法
 - 字段(field)：类的属性
 - 方法(method)：类的行为，与属性有关的功能和操作
 - 构造函数是一种特殊的方法，用来初始化类的一个新的实例
 - 构造函数具有和类名相同的名称，而且不返回任何数据类型

```
class Person {  
    String name;  
    int age;  
    void sayHello ()  
    {  
        System.out.println("Hello World");  
    }  
    Person(String n, int a)  
    {  
        name = n;  
        age = a;  
    }  
}
```

实例 (Instance)

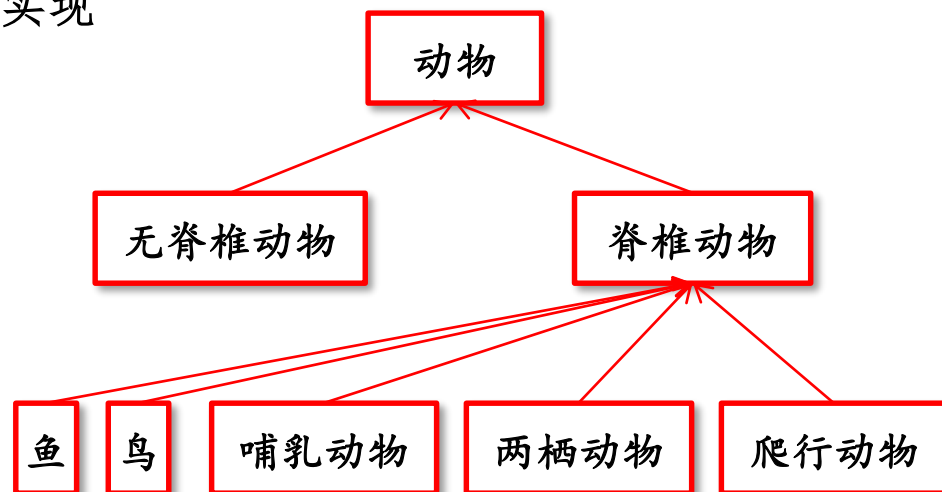
- 实例(instance)
 - 类的具体对象

```
class Person {  
    String name;  
    int age;  
    void sayHello ()  
    {  
        System.out.println("Hello World");  
    }  
    Person(String n, int a)  
    {  
        name = n;  
        age = a;  
    }  
}  
  
Person alice("Alice", 20);  
Alice.sayHello();
```

继承 (Inheritance)

- 现实世界中的类呈层级结构
- 继承(inheritance)是面向对象编程中对现实世界层级结构的实现
- 子类(subclass)的定义基于父类(superclass)
- 子类是对父类定义的扩展
- Java中的继承通过**extends**关键字来实现
- Java禁止多重继承

```
class Animal {  
    .....  
}  
class Vertebrata extends Animal {  
    .....  
}  
class Bird extends Vertebrata {  
    .....  
}
```



继承

- 字段的继承和添加
- 方法的继承和添加

```
class Person {  
    String name;  
    int age;  
    void sayHello () {  
        System.out.println ("Hello World");  
    }  
}
```

```
class Student extends Person {  
    String major; //添加的字段  
    Student (String n, int a, String m) { //子类构造方法  
        name = n; age = a; major = m;  
    }  
    void changeMajor (String m) { //添加的方法  
        major = m;  
    }  
}
```

```
Student alice ("Alice", 20, "Physics");  
alice.sayHello (); //调用从父类继承的方法  
alice.changeMajor ("Computer Sci."); //调用添加的方法
```

继承

- 上溯造型

- 子类对象可被视为其父类的一个对象

Person p = new Student ("Alice", 20, "Physics");

Animal a = new Dog ();

- 反之错误

Dog dog = new Animal();

字段的隐藏

- 子类重新定义父类的属性
- 父类属性被隐藏

```
class Person {  
    String name;  
    int age = 0; //父类属性age, 初始化为0  
    void sayHello () {  
        System.out.println ("Hello World");  
    }  
}  
  
class Student extends Person {  
    String major; //添加的字段  
    int age; //隐藏父类的属性age  
    Student (String n, int a, String m) {//子类构造方法  
        name = n; age = a; major = m;  
    }  
}  
  
Student alice ("Alice", 20, "Physics");  
System.out.println (alice.age); //will print 20
```

覆盖 (Overriding)

- 子类可以定义与父类成员名字相同的方法
- 是对父类方法的重定义
- 和重载(**overloading**)的区别
 - 覆盖：形式完全相同，是原有方法的重新实现
 - 重载：形式不同（参数类型、个数...），是原有方法的一种新的实现

```
class Person {
    String name;
    int age;
    void sayHello () {
        System.out.println ("Hello World");
    }
}

class Student extends Person {
    String major; //添加的字段
    Student (String n, int a, String m) { //子类构造方法
        name = n; age = a; major = m;
    }
    void sayHello () { //覆盖父类的sayHello
        System.out.println("Student say hello");
    }
    void sayHello (String word) { //重载父类的sayHello
        System.out.println(word);
    }
}

Student alice ("Alice", 20, "Physics");
alice.sayHello (); //will print "Student say hello"
alice.sayHello ("123"); //will print "123"
```

Java

- 类和实例
- 继承、隐藏和覆盖
- 多态
- 接口
- 包

多态

- 同名方法和属性的运行时绑定

```
class Animal {  
    int age = 0;  
    void eat ()  
    {  
        System.out.println ("Animal eat");  
    }  
}
```

```
class Dog extends Animal {  
    int age = 10;  
    void eat ()  
    {  
        System.out.println ("Dog eat");  
    }  
}
```

```
Animal a = new Dog ();  
System.out.println ( a.age ); // ??  
a.eat (); // ??
```

编译时类型和运行时类型

- 编译时类型：声明时使用的类型
- 运行时类型：指向的对象的类型
- 例如： `Animal a = new Dog();`
 - `a`的编译时类型为 **Animal**
 - `a`的运行时类型为 **Dog**

多态

- 同名方法和属性的运行时绑定
 - 被隐藏的属性，子类对象被转换成父类后，访问的是父类的属性。（由编译时类型决定）
 - 被覆盖的方法，子类对象被转换成父类后，调用的仍然是子类的方法。（由运行时类型决定）

```
class Animal {  
    int age = 0;  
    void eat ()  
    {  
        System.out.println ("Animal eat");  
    }  
}
```

```
class Dog extends Animal {  
    int age = 10;  
    void eat ()  
    {  
        System.out.println ("Dog eat");  
    }  
}
```

```
Animal a = new Dog ();  
System.out.println ( a.age ); // ??  
a.eat (); // ??
```



```
class Animal {  
    int age = 0;  
    void setAge (int a) {  
        age = a;  
    }  
    void printAge () {  
        System.out.println (age);  
    }  
    void printAnimalAge () {  
        System.out.println (age);  
    }  
}
```

```
class Dog extends Animal {  
    int age = 10;  
    void printDogAge () {  
        System.out.println (age);  
    }  
}
```

```
class Animal {  
    int age = 0;  
    void setAge (int a) {  
        age = a;  
    }  
    void printAge () {  
        System.out.println (age);  
    }  
    void printAnimalAge () {  
        System.out.println (age);  
    }  
}
```

```
class Dog extends Animal {  
    int age = 10;  
    void printDogAge () {  
        System.out.println (age);  
    }  
}
```

```
public class Test {  
    public static void main (String args[])  
    {  
        Dog dog;  
  
        dog = new Dog ();  
        dog.setAge (20);  
        dog.printDogAge ();  
        dog.printAnimalAge ();  
        dog.printAge ();  
        System.out.println (dog.age);  
    }  
}
```

```
class Animal {  
    int age = 0;  
    void setAge (int a) {  
        age = a;  
    }  
    void printAge () {  
        System.out.println (age);  
    }  
}
```

```
class Dog extends Animal {  
    int age = 10;  
    void setAge (int a) {  
        age = a;  
    }  
}
```

```
public class Test {  
    public static void main (String args[])  
    {  
        Animal animal;  
  
        animal = new Dog ();  
        animal.setAge (20);  
        animal.printAge ();  
        System.out.println (animal.age);  
    }  
}
```

- 若 `class B extends A`

<div>编译时类型</div> <div>运行时类型</div>	父类	子类
父类	<pre>A a = new A();</pre> 始终使用父类属性	<pre>B b = new A();</pre> 类型错误
子类	<pre>A a = new B();</pre> <ul style="list-style-type: none">• 调用父类方法时使用父类属性;• 调用子类方法时使用子类属性;	<pre>B b = new B();</pre> 始终使用子类属性

Java

- 类和实例
- 继承、隐藏和覆盖
- 多态
- 接口
- 包

理解Java接口

- 接口（**interface**）是一种抽象类型，用来定义多个类可能具有的相似行为

理解Java接口

- 接口（**interface**）是一种抽象类型，用来定义多个类可能具有的相似行为
- 接口可以表征不同类之间的相似行为，但不一定与这些类直接有直接关系
 - 人和鳄鱼都会流泪
 - 人和鳄鱼可以被定义为动物的子类，但不合适被定义为“流泪者”的子类...

理解Java接口

- 接口（**interface**）是一种抽象类型，用来定义多个类可能具有的相似行为
- 接口可以表征不同类之间的相似行为，但不一定与这些类直接有直接关系
 - 人和鳄鱼都会流泪
 - 人和鳄鱼可以被定义为动物的子类，但不合适被定义为“流泪者”的子类...
- 与**contract**和**protocol**的概念相似
 - 了解类的行为而不需理解具体实现
 - 麦当劳和肯德基

Interface和Class

- 相似点
 - 可以包含任意数量的方法
 - 需要被写入后缀为.java的文件, 文件名需要与接口名一致
 - 编译后成为独立的后缀为.class的字节码文件

Interface和Class

- 相似点
 - 可以包含任意数量的方法
 - 需要被写入后缀为.java的文件, 文件名需要与接口名一致
 - 编译后成为独立的后缀为.class的字节码文件
- 不同点
 - 接口不能被实例化
 - 没有构造函数
 - 只包括抽象或静态方法
 - 除static final外不包含其他类型的字段
 - 只包含常量
 - 接口允许多重继承, 而类不可以

接口的声明

```
[visibility] interface InterfaceName [extends other interfaces] {  
    constant declarations  
    abstract method declarations  
}
```

```
public interface Predator {  
    boolean chasePrey(Prey p);  
    void eatPrey(Prey p);  
}
```

- 默认情况下接口只对同一package中的类可见
- public接口对所有类可见
- 接口中的字段默认为 **static final** 型
 - 可以在申明中省略‘static final’关键字
- 接口中的方法默认为 **public abstract**
 - 可以在申明中省略‘public abstract’关键字

接口的实现

```
public class Lion implements Predator {  
    public boolean chasePrey(Prey p) {  
        // programing to have the lion chase the prey  
    }  
    public void eatPrey(Prey p) {  
        // programming to have the lion eat the prey  
    }  
}
```

- 当某个class实现一个interface时，相当于该class签署了一个合同，同意遵守接口所规范的行为
- 如果一个class没有全部实现interface中声明的方法，则该class必须声明为abstract class
- 一个class可以实现多个接口

```
public class Lion implements Predator, Prey { ... }
```

对接口的引用

- 接口可以用来声明引用类型的变量
- 该变量可指向实现该接口的类的实例
- 把接口作为一种数据类型可以不需要了解具体的类，而着重于接口定义的行为

// in file 'Printable.java'

```
interface Printable {  
    void print();  
}
```

// in file 'Printable.java'

```
interface Printable {  
    void print();  
}
```

// in file 'Hello.java'

```
public class Hello implements Printable{  
    void print() {  
        System.out.println("Hello");  
    }  
}
```

// in file 'Byebye.java'

```
public class Byebye implements Printable{  
    void print() {  
        System.out.println("Byebye");  
    }  
}
```

// in file 'Printable.java'

```
interface Printable {  
    void print();  
}
```

// in file 'Hello.java'

```
public class Hello implements Printable{  
    void print() {  
        System.out.println("Hello");  
    }  
}
```

// in file 'Byebye.java'

```
public class Byebye implements Printable{  
    void print() {  
        System.out.println("Byebye");  
    }  
}
```

// in file 'Main.java'

```
public class Main {  
    public static void main(String args[]) {  
        // declare an interface type variable  
        Printable p;  
    }  
}
```


// in file 'Printable.java'

```
interface Printable {  
    void print();  
}
```

// in file 'Hello.java'

```
public class Hello implements Printable{  
    void print() {  
        System.out.println("Hello");  
    }  
}
```

// in file 'Byebye.java'

```
public class Byebye implements Printable{  
    void print() {  
        System.out.println("Byebye");  
    }  
}
```

// in file 'Main.java'

```
public class Main {  
    public static void main (String args[]) {  
        // declare an interface type variable  
        Printable p;  
        // point to the instance of Hello  
        p = new Hello();  
        // call the method defined by interface,  
        system will print 'Hello'  
        p.print();  
    }  
}
```

// in file 'Printable.java'

```
interface Printable {  
    void print();  
}
```

// in file 'Hello.java'

```
public class Hello implements Printable{  
    void print() {  
        System.out.println("Hello");  
    }  
}
```

// in file 'Byebye.java'

```
public class Byebye implements Printable{  
    void print() {  
        System.out.println("Byebye");  
    }  
}
```

// in file 'Main.java'

```
public class Main {  
    public static void main (String args[]) {  
        // declare an interface type variable  
        Printable p;  
        // point to the instance of Hello  
        p = new Hello();  
        // call the method defined by interface,  
        system will print 'Hello'  
        p.print();  
        // now point the instance of Byebye  
        p = new Byebye();  
        // system will print 'Byebye'  
        p.print();  
    }  
}
```

Java

- 类和实例
- 继承、隐藏和覆盖
- 多态
- 接口
- 包

包 (package)

- **Java**中每个类都属于一个包，包是一个松散的类的集合
- 一般不要求处于同一个包中的类有明确的相互关系，如包含、继承
- 但是由于同一包中的类在默认情况下可以互相访问，所以为了方便编程和管理，通常把需要在一起工作的类放在一个包里
- 简单来看，**Java**包就是文件目录
- 如果要使用其他包里的类，则需要预先导入 (**import**)

定义和使用包

- 定义

```
package path.to.package.foo;  
class Foo {  
    ...  
}
```

- 使用

```
import path.to.package.foo.Foo;  
import path.to.package.foo.*;
```

package语句

- `package pkg1[.pkg2[.pkg3...]];`
- 包及子包的定义，实际上是为了解决名字空间、名字冲突，它与类的继承没有关系。事实上，一个子类与其父类可以位于不同的包中
- Java的JDK提供的包包括：`java.applet`，`java.awt`，`java.awt.image`，`java.awt.peer`，`java.io`，`java.lang`，`java.net`，`java.util`，`javax.swing`，等
- 包层次的根目录是由环境变量CLASSPATH来确定的。
- 在简单情况下，Java源文件默认为package语句，这时称为无名包（`unnamed package`）。无名包不能有子包

import语句

- 为了能使用Java中已提供的类，需要用import语句来引入所需要的类。
import语句的格式为：
- `import package1[.package2...]. (classname | *);`
- 其中，`package1[.package2...]`表明包的层次，与package语句相同，它对应于文件目录，`classname`则指明所要引入的类，如果要从一个包中引入多个类，则可以用星号(*)来代替。例如：

```
import java.awt.*;  
import java.util.Date;
```
- Java编译器为所有程序自动引入包java.lang，因此不必用import语句引入它包含的所有的类，但是若需要使用其他包中的类，必须用import语句引入。
- 注意：使用星号(*)只能表示本层次的所有类，不包括子层次下的类。
- 例如，经常需要用两条import语句来引入两个层次的类：

```
import java.awt.*;  
import java.awt.event.*;
```

About Mini Java

基本特性

- MiniJava 是 Java 的子集，缺省约定遵从 Java
- 不允许方法重载(Overloading)
 - 不通过不同的参数来区分方法实现体，以简化编译器的实现
 - 但子类的方法自动覆盖父类定义的方法（多态）
 - 子类的属性也自动隐藏父类定义的属性
- 类中只能申明变量和方法（不能嵌套类）
- 只有类，没有接口，有继承关系（单继承）
- 不支持注释

基本特性

- 一个文件中可以声明若干个类
有且只有一个主类，辅类可以有多个，类不能申明为public;
- 表达式(Expression)一共有9种
加、减、乘、与、小于、数组定位、数组长度、消息传递
(即参数传递)、基本表达式
 - 简化掉的操作符：“/”,“||”,“==”,“>”等
 - 不支持“连加”等“: “a+b+c”, “a*b-c”
- 基本表达式(Primary Expression)一共有9种
整数(Integer)、“真”(true)、“假”(false)、对象、
this、初始化(allocation)、数组初始化(array allocation)、
非(not)、括号(bracket)