学些参考网址：https://www.shiyanlou.com/courses/running

文档和笔记

**认识 Java**

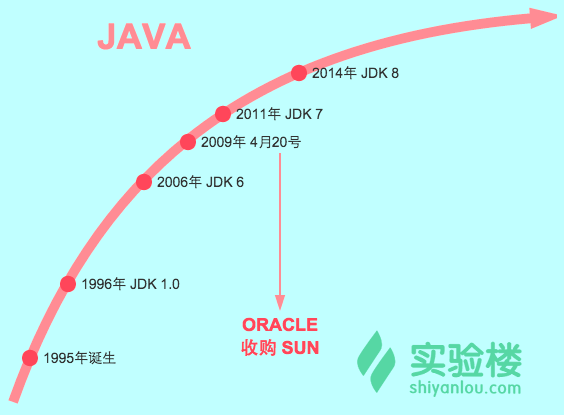
**实验介绍**

　　本实验将介绍 Java 平台应用和 Java 的核心概念：JVM（java虚拟机，java virtual machine）、JDK(java development kit,java开发包)、JRE（java运行时环境，java runtime environment）。同时我们会学到如何在 windows 系统中搭建 Java 开发环境，以及如何利用第三方工具进行 Java 程序的开发。并且在这一章里我们会先接触 Java 的第一个程序 Hello World！让我们一起进入 Java 语言的神奇世界吧！

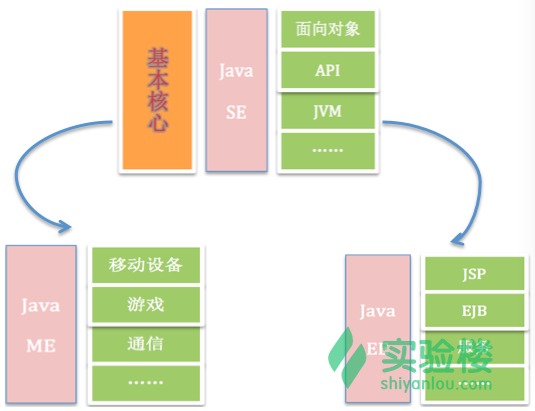
　　由于本平台的环境在 Linux 系统下已经为同学们搭建好了，下面的搭建过程主要是讲解在 Windows 系统下的搭建方法，同学们可以在自己的电脑上自行操作。

**一、Java简介**

　　Java 是一种面向对象的语言。于1995年，由 Sun 公司出品。虽然 Java 仅仅只产生了短短20年，但是它的发展是非常迅速的。在2009年4月20号，ORACLE 收购了 Sun 公司，也就是说 Java 这门语言现在归属于 ORACLE 这家公司门下。



　　在 Java 这门语言体系当中，最基础的部分就是 Java SE(也叫j2se，java standard edition) 部分，Java 的标准版本。它包括Java最基础的一些结构，包括面向对象的一些特性等等，同时它也是 Java 技术基础和核心。在 Java SE 的基础之上，又分为了 Java EE（Java 的企业版），应用于大型企业级应用的开发。Java ME 主要用于嵌入式开发。初学的时候我们都是从 Java SE 开始的。



　　JVM （java virtual machine）叫 Java 虚拟机，它也是整个 Java 技术的核心。Java 语言的跨平台就多亏了 JVM—java程序的运行不是在宿主机上而是在jvm虚拟机里面运行，而只要jvm能够在宿主机上运行那么java程序就能执行。

　　JDK (java development kit)叫 Java 开发工具包，没有 JDK 就没有办法对 Java 程序的开发。

　　JRE(java runtime environment) 叫 Java 运行环境，如果我们需要运行一个Java程序，就得安装 JRE，运行java程序需要jre，而不需要jdk。

　　JDK、JRE和JVM之间的关系：



jdk开发java程序，jvm和jre负责运行程序。

**二、Java开发环境搭建**

下载

安装

配置JAVA\_HOME；CLASSPATH

Path

搭建Java开发环境，第一步我们就需要安装JDK。大家可以在ORACLE官方网站上下载，下载路径：

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

　　这里要提醒各位同学，不同平台和系统的安装文件是不一样的，根据自己电脑的情况选着合适的版本进行安装。

　　接下来我们就来讲讲 windows 系统下 JDK 的安装。

　　第一步：安装JDK

　　在 window 系统下，下载JDK后，点开安装程序，按照指示按下一步就可以了。

　　第二步：配置环境变量

　　当我们安装好JDK后，就要配置环境变量了。一般来说，我们会配置这三个环境变量：

* JAVA\_HOME 配置 JDK 安装路径，jdk的安装根目录
* PATH 配置 JDK 命令文件的位置，jdk的bin目录
* CLASSPATH 配置类库文件的位置，jdk的lib目录

　　步骤一：右击计算机，选着属性

　　步骤二：打开高级系统设置

　　步骤三：选择高级

　　步骤四：选择环境变量

　　步骤五：在系统变量里配置JAVA\_HOME

　　　　1. 点击系统变量下的新建

　　　　2. 变量名填JAVA\_HOME

　　　　3. 变量值填JDK的安装路径（打开计算机->C盘->Program Files->Java->jdl1.7.0.13(或者其他版本)，拷贝路径就可以了）

　　　　4. 点击确定

　　步骤六：在在系统变量里配置PATH变量

　　　　1. 系统变量默认里有Path变量，找到它，点击编辑

　　　　2. 在变量值最前面添加指定jdk命令文件的位置（jdk1.7.0.13->bin目录，拷贝路径）

　　　　3. 将路径添加到Path变量值前面用;和后面的路径隔开

　　步骤七:配置CLASSPATH变量

　　　　1. 点击系统变量下的新建

　　　　2. 变量名填CLASSPATH

　　　　3. 变量值填JDK的安装路径（打开计算机->C盘->Program Files->Java->jdl1.7.0.13(或者其他版本)->lib目录，拷贝路径就可以了）

　　　　4. 点击确定

　　第三步：验证环境配置是否正确

　　1. 点击开始

　　2. 在搜索中输入 cmd，按回车，打开 dos 命令行

　　3. 输入命令 java ，按回车

　　4. 下面会出现一系列的命令行，这些提示是Java工具的用法和作用

　　5. 再执行一个命令，输入javac，按回车

　　6. 如果以上均给出提示，则开发环境配置是没有问题的

7. 同学们也可以在右边的桌面上打开Xfce 终端，输入以上验证操作命令，进行观察。

安装jdk三步：

1. jdk下载安装
2. jdk环境变量配置:
3. JAVA\_HOME：jdk的安装路径/your/~/path/java-1.0.1
4. PATH:jdk的命令执行路径/your/~/path/java-1.0.1/bin
5. CLASSPATH:jdk类库文件的路径/your/~/path/java-1.0.1/lib
6. 安装配置验证:java –v javac -v

**三、利用文本文件编辑 Java 程序**

Java源文件

Javac 源文件生成字节码文件

Java 类名执行程序

Java也是解释执行，由jvm来解释字节码

　我们如何用文本文件去开发一个Java程序呢？主要有三步。

　　第一步，我们使用文本文件去编写Java的源代码文件，Java的源代码文件以.java结尾，源代码文件默认是不能被机器执行的(机器执行的是二进制码)。

　　第二步，我们要使用编译器（也就是javac命令），对源代码文件进行编译的操作。把它编译成字节码文件，字节码文件是以.class结尾。它与平台无关，也就是说无论你是windows还是linux，字节码文件都是这样的一个文件，这跟系统没有关系。实际上Java的跨平台就是因为字节码文件的存在。

第三步，就是我们的解释器。针对不同的平台，根据相应的解释器，去解释运行字节码文件（这里我们使用java命令），最终我们可以看到程序运行出的结果。

源代码\*\*.java 🡪编译器javac \*\*.java,得到字节码文件\*\*.class🡪解释器java \*\*

大家在右侧的桌面试试吧：

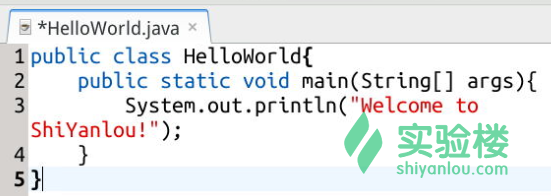
　　1. 在右侧桌面点击右键，选择从模板创建，点击空文件。



　　2. 将文件命名为HelloWorld.java,点击创建。

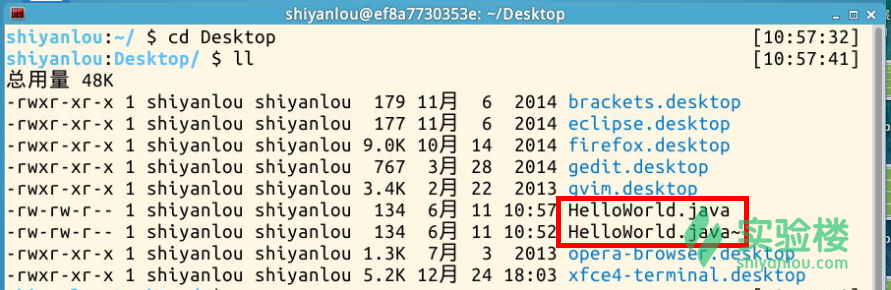
　　3. 双击点开桌面上我们刚刚建好的文件，进行编辑

　　4. 看看这是木木在上面书写的代码，同学们可以参照到下面图片里的代码进行输入。至于这些代码的含义，我们等会儿便会讲到。



　　5. 代码编辑完了保存一下（可以按ctrl＋s或在菜单栏的file里选择save）

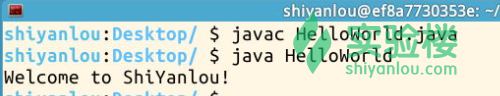
　　6. 点开Xfce 终端，输入cd Desktop,因为我们将文件存放在桌面上的，所以我们要把目录定位到桌面（这时候可以输入ll,来查看一下文件是否在桌面上）



　　7. 输入javac HelloWorld.java,对源文件进行编译

　　8. 同学们是不是发现桌面上多出了一个HelloWorld.class的文件

　　9. 接着我们继续输入java HelloWorld（java 命令不能跟上文件的后缀名）,对HelloWorld.class字节码文件利用解释器去执行它，是不是出现了Welcome to ShiYanlou!



**四、Eclipse**

Eclipse（或 MyEclipse）是一个 IDE 工具，IDE 是集成开发环境的意思。它将程序开发环境和程序调试环境集合在一起，提高开发效率。我们在做开始的时候，一般都会借助这种第三方的工具，给大家提供 Eclipse 的官方下载地址：<http://www.eclipse.org/downloads/>

Eclipse 它是一款开源软件，所以是免费的。

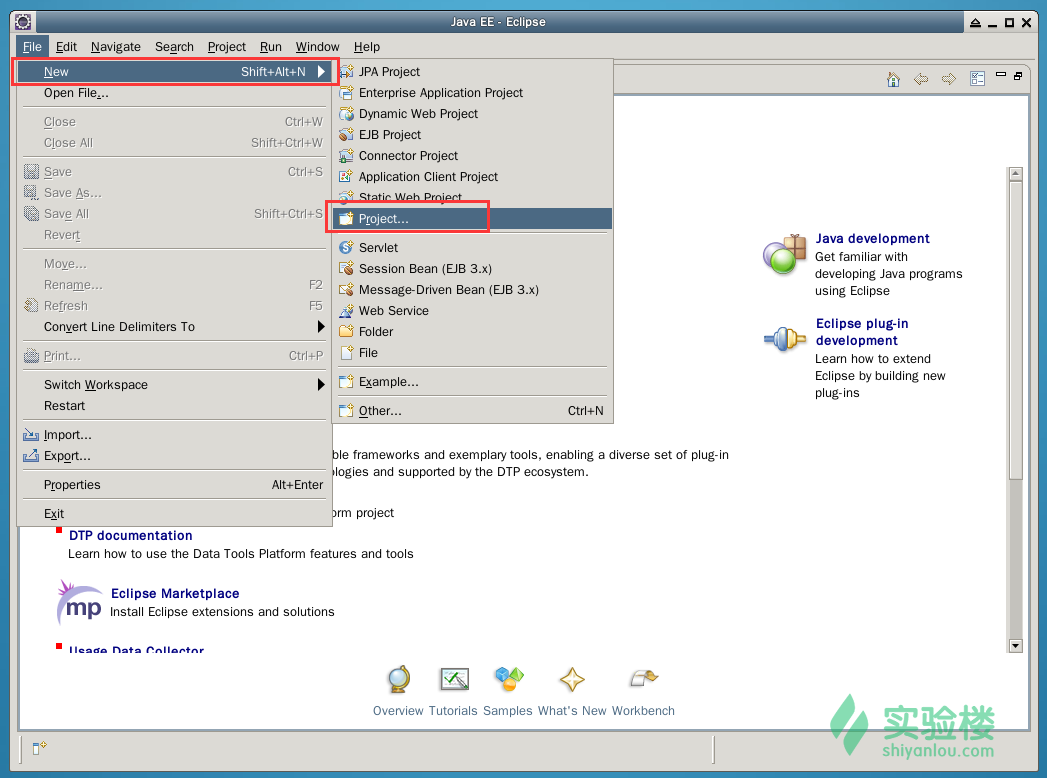
同学们请打开桌面上的 Eclipse，在 Eclipse 上开发 Java 程序一共分为四个步骤：

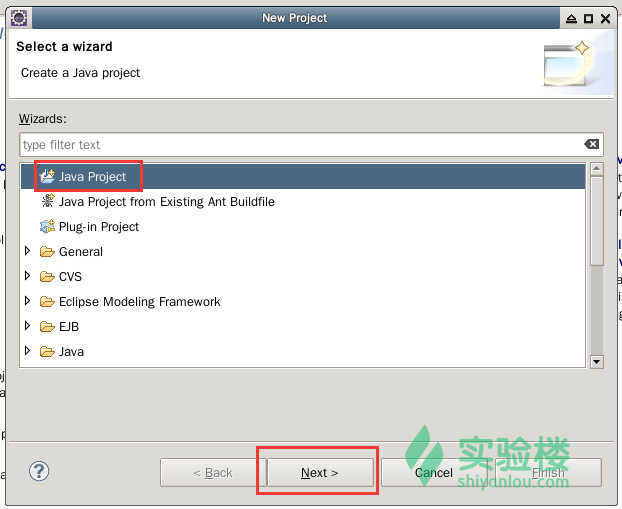
1. 创建一个 Java 项目
2. 创建程序包
3. 编写 Java 源程序
4. 运行 Java 程序

接下来我们就来创建一个Java项目吧！

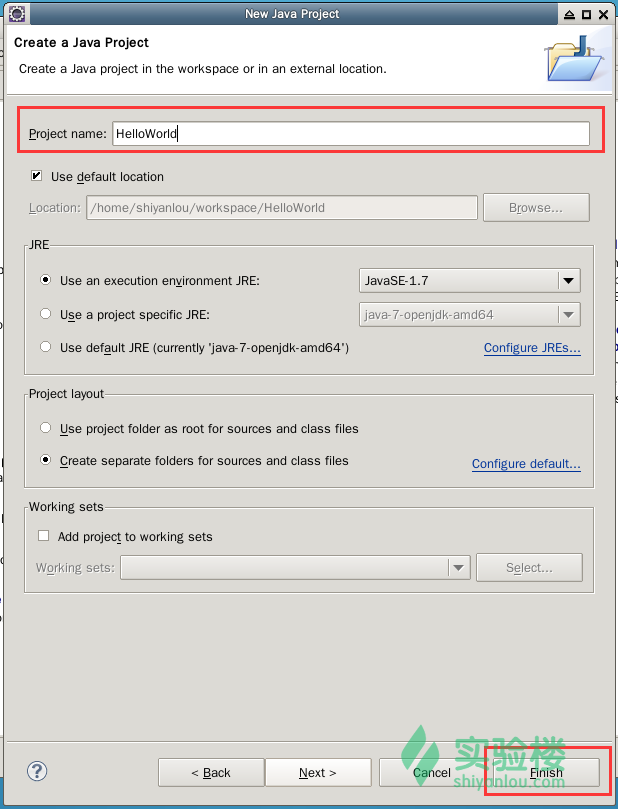
1.双击 Eclipse 进入我们的开发环境。第一次进入的时候，Eclipse 可能会让你设置工作空间地址(Workspace），直接点OK就可以了。在实验楼的环境中，Workspace 的默认地址为 /home/shiyanlou/workspace。

2.等待Eclipse启动完成，点击菜单里中的 File ，选择 New ，创建一个 Project ,选择 Java Project 新建一个Java工程（同时同学们也可以右击左侧空白地方，选择 New ，创建 Project ，选择Java Project）。

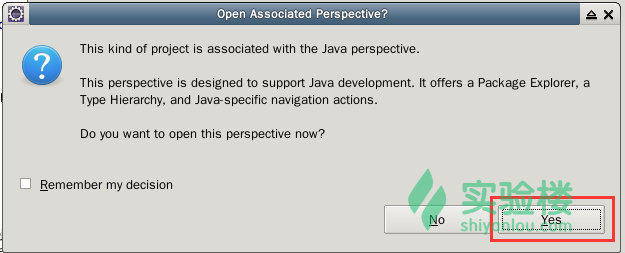




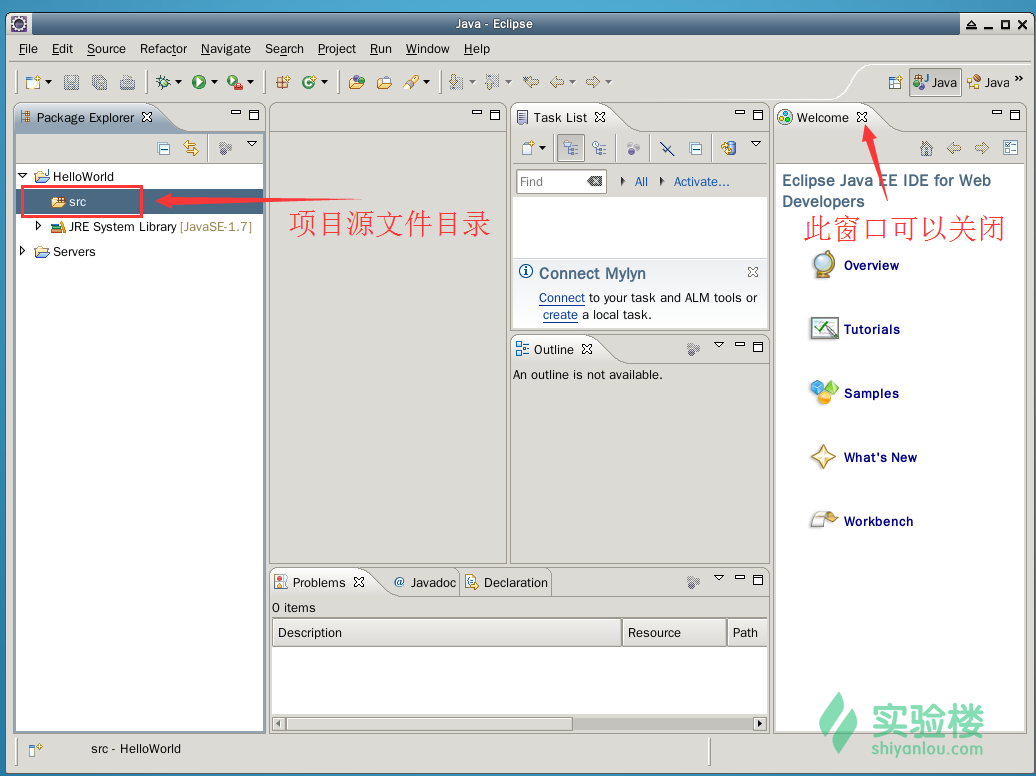
3.接下来我们要为我们的工程命名，比如说指定一个名称 HelloWorld 。



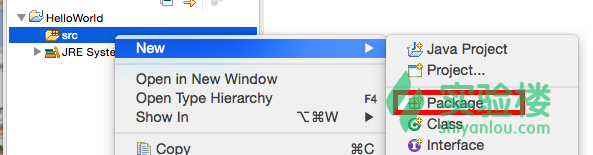
点击 Finish 按钮完成创建，如果遇到下面的对话框，点击 Yes即可。



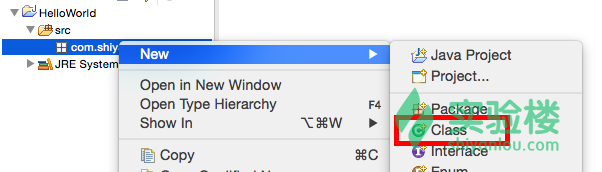
4.创建好了工程，点开工程发现有许多下拉文件，src用来放置我们的源代码文件。



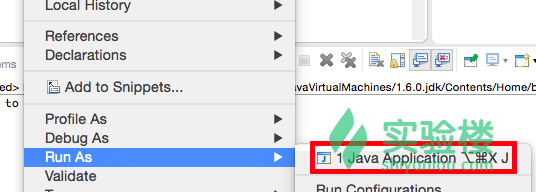
5.接下来我们要创建程序包，创建包的目的是便于我们对源代码的管理。我们可以将不同的源代码文件放到指定的包下，避免重名问题。右击src,选择New,创建一个Package，指定包名（一般为域名的反向）。点击finish完成。在src的下拉目录里就有了我们定义的包。



6.接下来我们便在指定的包下面来编写源代码。右击我们刚刚创建的包，选择New，新建一个Class源代码文件，指定类的名称，我们依然取成HelloWorld,点击finish,我们就可以看到，在我们的包下有一个HelloWorld.java的文件，接下来就在我们的源代码文件里输入我们上面的代码吧。



7.接下来就进入了第四步，运行Java程序。右击空白位置，选择Run As,运行Java Application。



8.于是我们便会在下面的窗口中看到Welcome to ShiYanlou!的输出。

程序编写基本步骤：

新建工程🡪在src（源代码文件存放处）新建程序包🡪在指定的包下编写程序文件🡪run as 执行。

**五、Hello World**

今天我们输入了如下代码两次：

public class HelloWorld

{

public static void main(String[] args)

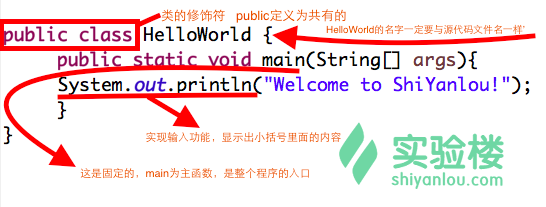
{

System.out.println("Hello World!");

}

}

　　那这些代码究竟是什么意思呢？



类名和文件名要一致，包括大小写。

**Java 基本语法**

**实验介绍**

　　在本章我们就来学习一下 Java 基本语法里的关键字、标识符、注释、变量和常量的相关知识。

**一、开发第一个 Java 程序**

　　还记得上一节课的 Hello World 吗？这一节课，我们还是先来看一看这一段代码。

public class HelloWorld

{

public static void main(String[] args)

{

System.out.println("Hello World!");

}

}

上面程序中包括 Java 的一些基本特征：

* 类(class)：上面程序定义了一个类 HelloWorld，该类的名字与 .java 文件的名字相同。（一个java文件中只能还有一个public修饰的类，且该类名和文件名必须相同。）
* 方法(method)：类的内部定义了该类的一个方法 ，其中main 被叫做主函数，是java程序的入口。
* 语句(statement)：“打印”这个功能是由一个语句实现，即：System.out.println("Hello World!");

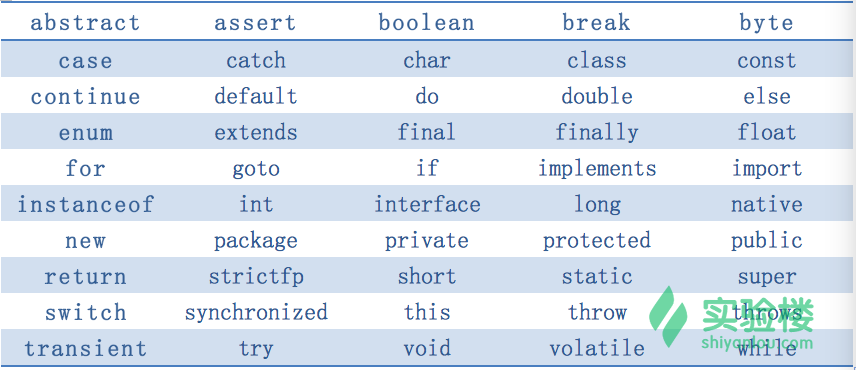
下面两点有关 Java 的书写方式：

* Java 中的语句要以;结尾 。
* 用花括号{}来整合语句，形成程序块。通过程序块，我们可以知道程序的不同部分的范围，比如类从哪里开始，到哪里结束。

**二、Java 关键字**

　　Java 的关键字对 java 的编译器有特殊的意义，他们用来表示一种数据类型，或者表示程序的结构等，关键字不能用作变量名、方法名、类名、包名。

　　Java 关键字有如下表所列，同学们先有个印象，具体含义我们将在后续的内容中详细讲解：



**三、Java 标识符**

　　Java 语言中，对于变量，常量，函数，语句块也有名字，我们统统称之为 Java 标识符.

　　标识符是用来给类、对象、方法、变量、接口和自定义数据类型命名的。

　　Java 标识符由数字，字母和下划线\_，美元符号$组成。在 Java 中是区分大小写的，而且还要求首位不能是数字。最重要的是，Java 关键字不能当作 Java 标识符。 下面的标识符是合法的：

shiYanlou, Shi\_Yanlou, $money, \_Java\_learning, OK, \_3th, \_3\_

String $str="this is a string";

下面的标识符是非法的：

#shiyan，25years，class，&name，if

在 Java 中，还有一些约定俗成的命名规则，希望同学们在写代码的时候都能遵循这些规则：

1. 类和接口名。每个字的首字母大写，含有大小写。例如，MyClass，HelloWorld，Time 等。
2. 方法名。首字符小写，其余的首字母大写，含大小写。尽量少用下划线。例如，myName，setTime 等。这种命名方法叫做驼峰式命名。
3. 常量名。基本数据类型的常量名使用全部大写字母，字与字之间用下划线分隔。对象常量可大小混写。例如，SIZE\_NAME。
4. 变量名。可大小写混写，首字符小写，字间分隔符用字的首字母大写。不用下划线，少用美元符号。给变量命名是尽量做到见名知意。
5. 命名过程中尽量做到见名知意，方便后期查看和修改代码，也方便其他人员的阅读。

**四、变量**

　　还记得我们在上面见到的 Java 关键字吗？在关键字里有一部分便是我们以后会经常用到的数据类型。通常我们在定义变量的时候，便会在变量名前面加上变量所对应的数据类型。

　　计算机语言通常需要在内存中存放数据，比如 C 语言中的变量，Java也有类似的变量。Java 和 C 语言都是静态类型的语言。在使用变量之前，要声明变量的类型。

变量(variable)占据一定的内存空间。不同类型的变量占据不同的大小。Java 中的变量类型如下：

| **数据类型** | **默认值** | **存储格式** | **数据范围** |
| --- | --- | --- | --- |
| short | 0 | 2个字节（-2^15~2^16-1） | -32,768到32767 |
| int | 0 | 4个字节(-2^31~2^31-1) | -2,147,483,648到2,147,483,647 |
| byte | 0 | 1个字节（8个bit即8个位）(-2^7~2^7-1) | -128到127 |
| char | /u0000 | 2个字节 | Unicode的字符范围 |
| long | 0L或0l | 8个字节 | -9,223,372,036,854,775,808到9,223,372,036, 854,775,807 |
| float | 0.0F或0.0f | 4个字节 | 32位IEEEE 754单精度范围 |
| double | 0.0或0.0D(d) | 8个字节 | 64位IEEE 754双精度范围 |
| boolean | false | 1位 | true(1)或false(0) |

　　在 Java 中，变量需要先声明(declare)才能使用。在声明中，说明变量的类型，赋予变量以特别名字，以便在后面的程序中调用它。你可以在程序中的任意位置声明变量。 比如:

public class Test

{

public static void main(String[] args)

{

System.out.println("Define a variable a is ");

int a; //声明变量a

a = 5;

System.out.println(a); // 打印一个整数a

}

}

　　上面 a 是变量名。可以在声明变量的同时，给变量赋值，比如 int a = 5;

　　变量的概念实际上来自于面向过程的编程语言。在Java中，所谓的变量实际上是基本类型 (premitive type)。我们将在类的讲解中更多深入。 上面的程序还可以看到，Java 中，可用//引领注释。

　　说到注释，在我们写代码的时候，为了方便理解和阅读，往往会在相关的代码附近添加说明文字，也就是我们的注释。注释是为了防止编译器在编译的时候将我们的注释也一并编译了，导致程序出错，所以在编译的时候，编译器会自动跳过我们编写的注释。

一般在 Java 里有三种注释：

* 行注释//：只注释一行
* 段注释/\*...\*/：注释若干行
* 文档注释/\*\*...\*/：注释若干行，并写入 javadoc文档

**五、自动类型转换和强制类型转换**

　　小伙伴们，我们运行一下如下的程序段，在控制台输出的结果会是怎样的呢？

public class HelloWorld{

public static void main(String[] args) {

double avg1=79.5;

int rise=10;

double avg2=avg1+rise;

System.out.println("调整前的平均分："+avg1);

System.out.println("调整后的平均分："+avg2);

}

}

　　在上面的程序中，我们定义了三个变量：avg1、rise、avg2，其中avg1是double类型的变量，而rise是int类型的变量，两种类型不一样的变量加在了一起变成了什么类型了呢？没错，在运算过程中rise被自动转换成了double类型的变量。

　　在Java程序中，不同的数据类型有些时候需要进行相互转换。数据类型转换就分为了自动类型转换和强制类型转换

　　自动类型转换是在程序执行过程中，不需要我们去特殊声明或者操作，变量由于需要而自动转换成了合适的数据类型。

自动类型转换需要满足下面的两个条件：

1. 目标类型与原类型兼容
2. 目标类型的字节数大于或等于原类型字节数（转换不会导致异常问题产生就能进行自动类型转换）

**六、常量**

主要指两种：字面常量和自定义常量。

有时候我们在写程序的时候，并不想因为某些操作而改变一些值，这时候我们就要用到常量这个概念了。所谓常量，我们也可以把它们理解为特殊的变量，只是它们在程序的运行过程中是不允许改变的。

一般常量有两种意思：

1. 第一种意思就是这个值本身，如一个整数9，我们就说是“一个int类型的常量9”。当然还有实数常量3.14，字符常量‘a’等等。
2. 另一种常量便表示不可变的变量，就是我们在变量声明前加上了 final 关键字。比如：final int i＝0；那么这个i的值便不能被修改。

**Java 运算符**

**实验介绍**

　　本实验将介绍 Java 的算术运算符、赋值运算符、逻辑运算符、比较运算符、条件运算符，以及运算符的优先级。

**一、算术运算符**

　　运算符顾名思义是一种符号，它是连接一个以上的操作符，实现某种功能的运算。

　　我们就先来认识一下算术运算符吧！

　　算术运算符主要实现的是算术运算，如我们常见的加法、减法、乘法、除法等。

| **算术运算符** | **名称** | **举例** | **类型** |
| --- | --- | --- | --- |
| + | 加法 | a = 6 + 7 | 双目运算符 |
| - | 减法 | a = 9 - 3 | 双目运算符 |
| \* | 乘法 | a = 8 \* 6 | 双目运算符 |
| / | 除法 | a = 4 / 2 | 双目运算符 |
| % | 取余 | a = 5 % 3 | 双目运算符 |
| ++ | 自增1 | ++i（或i++） | 单目运算符 |
| -- | 自减1 | --i（或i--） | 单目运算符 |

　　上面的例子中a和i均为一个变量，=为赋值运算符，功能便是将等式右边的运算结果赋给左边的变量。

取余操作就是求取余数的意思，如上面的例子中a = 5 % 3，由此可以知道a的值为2。

自加和自减运算有两种写法，虽然++i和i++，虽然它们都是自增1，但是根据++的顺序，++i是先加再取值，i++是先取值在加。

我们在 Eclipse 上来直观的感受一下吧！

第一步，我们先新建一个工程，然后新建一个包，再在包下新建一个源代码文件。（可参考第章一节,代码的类名要和源代码文件名一致）。

　　接下来输入以下代码，进行验证吧，同学们也可自行组织代码验证一下。

public class Test

{

public static void main(String[] args)

{

int a, b, c, d;

a = 6 + 7;

System.out.println("a = "+a);

b = a % 5;

System.out.println("b = "+b);

c = b++;

System.out.println("b++后 b = "+b);

System.out.println("c = "+c);

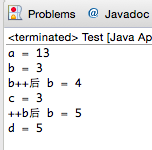
d = ++b;

System.out.println("++b后 b = "+b);

System.out.println("d = "+d);

}

}



　　大家可以发现++在变量的左边和右边，赋值得到的结果是不一样的，自减也是同样的道理。

**二、赋值运算符**

赋值运算符是指将右边的操作数或运算结果赋值给左边的操作数。

| **赋值运算符** | **名称** | **举例** | **类型** |
| --- | --- | --- | --- |
| = | 赋值 | a = c | 双目运算符 |
| += | 加等于 | a += c等价于a = a + c | 双目运算符 |
| -= | 减等于 | a -= c等价于a = a - c | 双目运算符 |
| \*= | 乘等于 | a \*= c等价于a = a \* c | 双目运算符 |
| /= | 除等于 | a /= c等价于a = a / c | 双目运算符 |
| %= | 取余等于 | a %= c等价于a = a % c | 双目运算符 |

练习：大家自己搭建一个项目，声明一些变量，将上面的赋值运算符进行一些操作，并将结果打印到控制台上。（同学们一定要自己动手练习哦，学习Java开发，实践才最重要，可以参见上面代码！）

**三、比较运算符**

　　比较运算符是用来判断两个操作数的大小的双目运算符，比较的结果是一个布尔值（true or false）。

java中常用的比较运算符如下：

| **比较运算符** | **名称** | **举例** | **结果** |
| --- | --- | --- | --- |
| > | 大于 | a = 1;b = 3; a > b | false |
| < | 小于 | a = 6;b = 9; a < b | true |
| >= | 大于等于 | a = 7; b = 8; a >= b | false |
| <= | 小于等于 | a = b; a <= b | true |
| == | 等于 | a = 8; b = 5; a == b | false |
| != | 不等于 | a = 8; b = 5; a != b | true |

注意：

1. >、<、>=、<=只支持左右两边的操作数是数值类型（即整形、浮点型等）。
2. ==、!=两边的操作数既可以是数值类型，又可以是引用类型（后面会讲什么叫引用）

大家来练习一下下面的代码吧！

public class Compare{

public static void main(String[] args) {

int a = 1;

double b = 7.2;

String str1 = "hello";

String str2 = "ShiYanlou";

System.out.println("a等于b：" + (a == b));

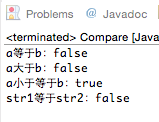
System.out.println("a大于b：" + (a > b));

System.out.println("a小于等于b：" + (a <= b));

System.out.println("str1等于str2：" + (str1 == str2));

}

}



**四、逻辑运算符**

　　逻辑运算符是通过运算符将操作数或等式进行逻辑判断的语句。是不是晕了，不过没关系，木木举个例子大家就懂了。不过我们先看看有哪些逻辑预算符吧！

| **逻辑运算符** | **名称** | **举例** | **类型** |
| --- | --- | --- | --- |
| && | 与 | a && c | 双目运算符 |
| ｜｜ | 或 | a ｜｜ c | 双目运算符 |
| ！ | 非 | !a | 单目运算符 |
| ^ | 异或 | a ^ c | 双目运算符 |

我们可以通过“举手表决”来理解逻辑运算符

* 与&&：a同意了（值为true），c也同意了（值为true），那这件事情肯定就Ok了，所以 a && c就为ture；但是其中有一个人不同意，这件事情不行，也就是a和c中，有一个为false， a && c就为false。
* 或||：这件事情只要有一个人同意了，就可以了。所以a和c中，有一个为true， a || c就为true。
* 非!：某个人原本同意，但非运算使他的结果变为相反的。a为true，!a就是false；a为false，!a就是true。
* 异或^：两个人的表决结果相同，就不通过；表决结果不同，才通过。所以a和c中，一真一假结果为真，其他的都是假。

看看如下的代码，同学们用逻辑运算符随意组合，看看输出的结果吧！

public class Logic {

public static void main(String[] args) {

boolean a = true; // a同意

boolean b = false; // b反对

boolean c = false; // c反对

boolean d = true; // d同意

System.out.println(a&&b);

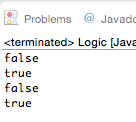
System.out.println((a||b));

System.out.println((!a));

System.out.println((c^d));

}

}



**五、条件运算符**

　　同学们从上面我们可以发现，我们把各类运算符分为了单目运算符和双目运算符，这是什么意思呢？单目运算符（或一元运算符）是只需要有一个操作数的运算符，双目运算符（或二元运算符）是需要两个操作数的运算符。

　　那有没有三目运算符呢？

　　条件运算符?:便是唯一的一个三目运算符了。

　　语法形式：布尔表达式？表达式1 : 表达式2

　　运算过程：如果布尔表达式的值为 true ，则返回 表达式1 的值，否则返回 表达式2 的值。

　　例如：

　　String str = (3<2） ? "3小于2" : "3大于2";

　　System.out.println(str);

　　分析一下上面的代码，很明显3<2是false,所以返回第二个表达式，即str = "3大于2"。

练习一下吧：

　　如果成绩大于或等于60 ，在控制台上输出及格，反之输出不及格，成绩用 score 表示，它的值定为68.9。

　　我们来分析一下首先我们得创建一个项目和源代码文件（当然你也可以先创一个包，再创建源代码文件），然后输入我们的固定结构（类和主函数）。

　　其次我们要定义一个变量 score，因为它的值为68.9，所以我们定义为 double。

　　然后我们利用条件表达式进行判断，最后再输出到控制台。

参考代码如下：

public class Test{

public static void main(String[] args) {

double score = 68.9;

String str = (score >= 60) ? "及格" : "不及格";

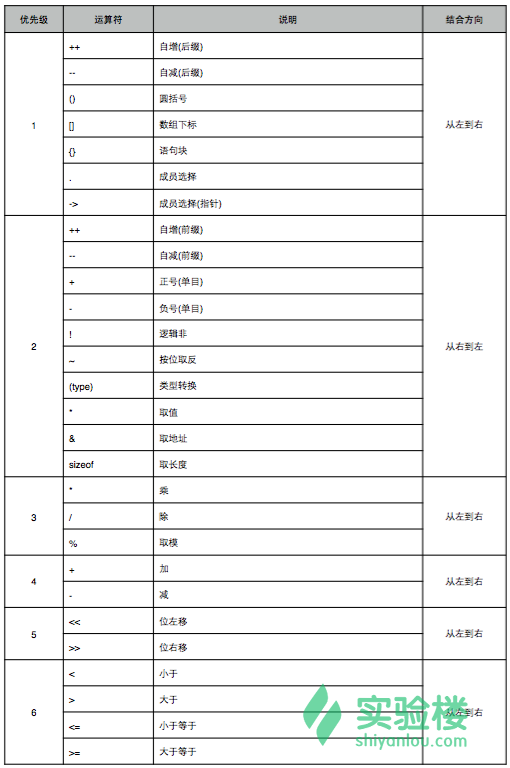
System.out.println("考试成绩："+str);

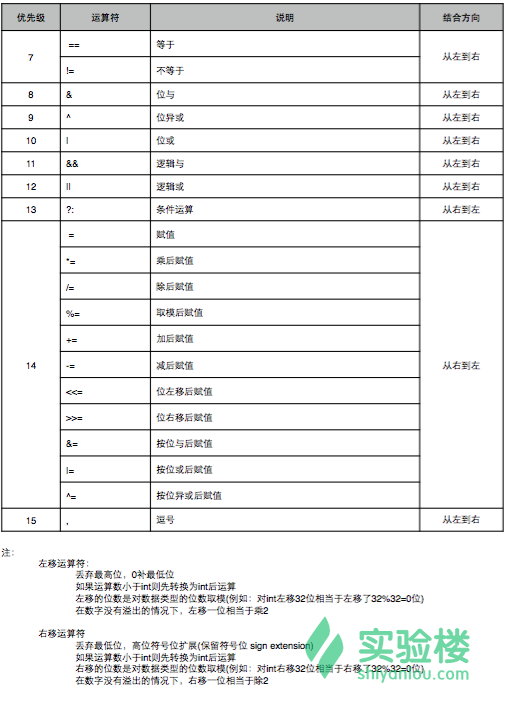
}

}

**六、运算符的优先级**

　　运算符的优先级是帮助我们在一个表达式中如何对于不同的运算符和相同的运算符，进行正确的运算顺序。

　　运算符的优先级大家在日常生活中通过多多运用自然就熟悉了，不需要特别地去记忆它，下面是木木多年的总结，有些运算符大家可能还认识，先熟悉一下就行了。 　　 !



**注：位移运算**

左移运算符：<<

丢弃最高位，0补最低位

如果运算数小于int则先转换为int后运算

左移的位数是对数据类型的位数取模（例如：对int—4字节32位—左移32为相当于左移了32%32=0位）

在数字没有溢出的情况下左移一位相当于乘以2

右移运算符：>>

丢弃最低位，高位符号位扩展（保留符号位sign extension），即以符号位来填充

如果运算数小于int则先转换为int后运算

右移的位数是对数据类型的位数取模（例如：对int 右移32为相当于右移了32%32=0位）

在数字没有溢出的情况下，右移一位相当于除以2

Java中的数字都是有符号的

Java还有无符号位运算：<<<和>>>

**Java 控制语句**

**实验介绍**

　　本实验将学习条件控制语句和循环控制语句相关的语法知识。

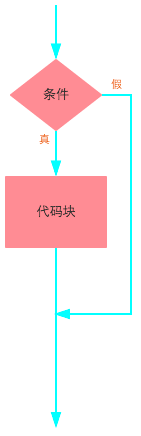
**一、条件控制语句**

语法：

if(条件){

条件成立时执行的代码

}



　　if...else 语句当条件成立时，则执行 if 部分的代码块； 条件不成立时，则进入 else 部分。例如，如果一个月天数大于30天，则为大月，否则为小月。

语法：

if(条件){

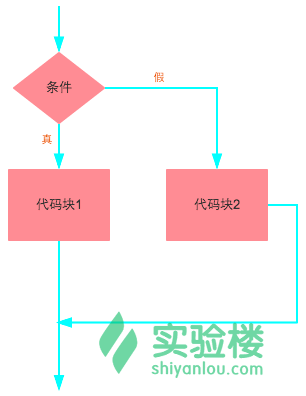
代码块1

}

else{

代码块2

}



　　多重 if 语句，在条件 1 不满足的情况下，才会进行条件 2 的判断，以此向下；当前面的条件均不成立时，最终执行 else 块内的代码。

语法：

if(条件1){

代码块1

}

else if(条件2){

代码块2

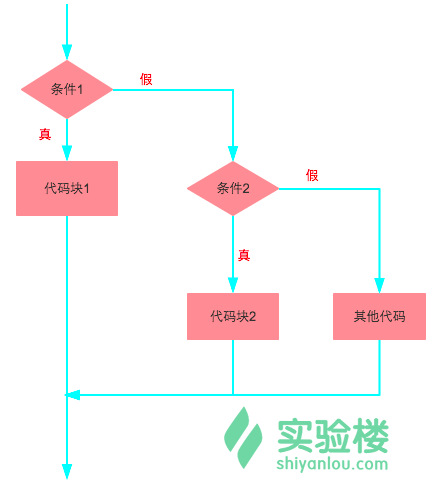
}

...

else {

代码块n

}



注意：如果 if(或 else if，或 else)条件成立时的执行语句只有一条，是可以省略大括号的！但如果执行语句有多条，那么大括号就是不可或缺的。

比如：

int days = 31;

if(days > 30)

System.out.println("本月是大月");

else

System.out.println("本月是小月");

　　if 语句是可以在内层进行嵌套的。嵌套 if 语句，只有当外层 if 的条件成立时，才会判断内层 if 的条件。

语法：

if(条件1){

if(条件2){

代码块1

}

else{

代码块2

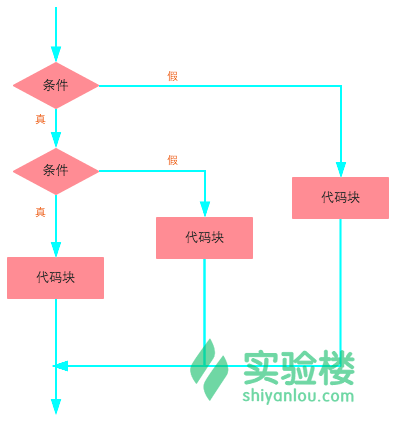
}

}

else{

代码块3

}



　　接下来我们来做一个简单的练习吧：小明考了78分，60分以上及格，80分以上为良好，90分以上为优秀，60分以下要重考，编写一段程序，输出小明的情况吧。

参考代码如下：

public class ScoreJudge {

public static void main(String[] args){

int score = 78;

if(score >= 60){

if(score >= 80){

if(score >= 90){

System.out.println("成绩优秀");

}

else{

System.out.println("成绩良好");

}

}

else{

System.out.println("成绩及格");

}

}

else{

System.out.println("需要补考");

}

}

}

　　上面木木给大家详细地介绍了 if 语句，接下来我们还要学习另一种条件语句：switch 语句。当需要对选项进行等值判断时，使用 switch 语句更加简洁明了。比如：摇号摇到1的得一等奖，摇到2的得二等奖，摇到3的等三等奖，摇到其他的没有奖。

语法：

switch(表达式){

case 值1:

代码块1

break;

case 值2:

代码块2

break;

...

default:

默认执行的代码块

}

　　当 switch 后表达式的值和 case 语句后的值相同时，从该位置开始向下执行，直到遇到 break 语句或者 switch 语句块结束；如果没有匹配的 case 语句则执行 default 块的代码。

例如上面的例子：

public class Draw {

public static void main(String[] args){

int num = 2;

switch(num){

case 1:

System.out.println("恭喜你，获得了一等奖");

break;

case 2:

System.out.println("恭喜你，获得了二等奖");

break;

case 3:

System.out.println("恭喜你，获得了三等奖");

break;

default:

System.out.println("很遗憾，下次再来");

}

}

}

**二、循环控制语句**

　　在我们的日常生活中，为了完成某些事情，我们需要重复地执行某些动作，比如我们走路的时候，要左右脚不停的重复交替。在计算机中，对一些重复的事情，是不是需要我们不停的输入呢？其实有一个更好的办法，循环语句。

　　我们先来学习一下循环语句中的while和do...while。

while语法：

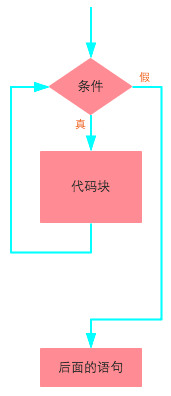
while(条件){

代码块

}

while 的执行过程是先判断，再执行。

1. 判断 while 后面的条件是否成立( true or false )
2. 当条件成立时，执行循环内的代码，然后重复执行1.、2.， 直到循环条件不成立为止



do...while语法：

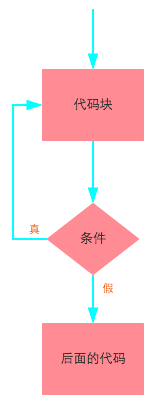
do{

代码块

}while(条件);

do...while 的执行过程是先执行，在判断（所以循环内的代码至少会执行一次）

1. 先执行一遍循环操作，然后判断循环条件是否成立
2. 如果条件成立，继续执行1.、2.，直到循环条件不成立为止



下面的例子：

int i = 0;

while(i < 100){

System.out.println("I love ShiYanlou!");

i++;

}

int i = 0;

do {

System.out.println("I love ShiYanlou!");

i++;

} while(i < 100);

　　接下来我们就来练习一下吧：分别用 while 和 do...while 两种方法，编写一个程序，实现 1-1000 中所有偶数的和，并输出。验证一下两种方法你输出的结果是一致吗？

参考如下：

public class SumOfOdd {

public static void main(String[] args){

int i1 = 1, i2 = 1;

int sum1 = 0, sum2 = 0;

while (i1 <= 1000){ //循环1000次

if(0 == i1 % 2){ //判断是否为偶数

sum1 += i1; //将偶数加入到总数里

}

i1++; //i自增1

}

System.out.println("用while，1到1000中，所有偶数的和为："+sum1);

do {

if (0 == i2 % 2){ //在条件语句中，将数值写在前面是为了防止将==写成了=

sum2 += i2;

}

i2++;

} while(i2 <= 1000);

System.out.println("用do...while，1到1000中，所有偶数的和为："+sum2);

}

}

　　最后我们来学习另一个循环语句for 语句

语法：

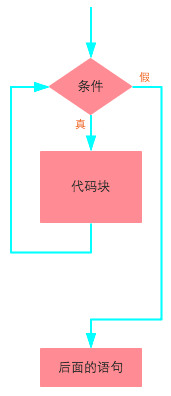
for(循环变量初始化; 循环条件; 循环变量变化){

循环操作

}

for 相比 while 和 do...while 语句结构更加简洁易读，它的执行顺序：

1. 执行循环变量初始化部分，设置循环的初始状态，此部分在整个循环中只执行一次
2. 进行循环条件的判断，如果条件为 true，则执行循环体内代码；如果为 false ，则直接退出循环
3. 执行循环变量变化部分，改变循环变量的值，以便进行下一次条件判断
4. 依次重新执行2.、3.、4.，直到退出循环



例如，计算100以内不能被整除的数之和：

int sum = 0; // 保存不能被3整除的数之和

// 循环变量 i 初始值为 1 ,每执行一次对变量加 1，只要小于等于 100 就重复执行循环

for (int i = 1;i<=100;i++) {

// 变量 i 与 3 进行求模（取余），如果不等于 0 ，则表示不能被 3 整除

if (i % 3 != 0) {

sum = sum + i; // 累加求和

}

}

System.out.println("1到100之间不能被3整除的数之和为：" + sum);

我们来练习一下吧，用 for 语句实现上面的练习题目。

参考代码如下：

public class SumOfOdd {

public static void main(String[] args){

int sum = 0;

for(int i = 1; i <= 1000; i++){

if(0 == i % 2){

sum += i;

}

}

System.out.println("用for，1到1000中，所有偶数和为："+sum);

}

}

　　　在讲条件控制语句的时候，给同学们讲解了 if 语句的嵌套，在循环语句里，三种循环语句可以自身嵌套，也可以相互嵌套，最常见的就是二重循环。在二重循环中，外层循环每执行一次，内层循环要执行一圈。

比如我想打印

\*

\*\*

\*\*\*

\*\*\*\*

这样的图形

// 外层循环控制行数

for (int i = 1; i<=4; i++ ) {

// 内层循环控制每行的\*号数

// 内层循环变量的最大值和外层循环变量的值相等

for (int j = 1; j<=i; j++ ) {

System.out.print("\*");

}

// 每打印完一行后进行换行

System.out.println();

}

练习一下：将这样的图形打印在控制台上

1

12

123

1234

12345

参考代码如下：

public class PrintNum {

public static void main(String[] args){

// 外层循环控制行数

for (int i = 1; i<=5; i++ ) {

// 内层循环控制每行的数

// 内层循环变量的最大值和外层循环变量的值相等

for (int j = 1; j<=i; j++ ) {

System.out.print(j);

}

// 每打印完一行后进行换行

System.out.println();

}

}

}

**三、跳转语句**

　　我们上面讲解 switch 语句的时候，看到了break这个关键字，它有什么用呢？同学们，我们将上面 switch 练习题里的代码中break去掉，运行一下，发现了什么问题吗？break的意思为跳出，经常用在条件和循环语句中，用来跳出***循环语句***的。

例如：

for(int i = 1; i <= 10; i++){

if(0 == i % 3){

break;

}

if(0 == i % 5){

System.out.println("我进来了！")

}

}

　　上面的代码中，本来 for 语句会使得循环体内的代码循环10次，但当i=3时,进入了第一个条件语句，遇到了break,结束了循环，而永远也进不了第二个条件语句。

　　在跳转语句中，还有一个continue跟break很像，它的作用是跳过循环体中剩余的语句执行下一次循环。

比如我们要打印10以内的所有奇数：

for(int i = 1; i <= 10; i++){

if(0 == i % 2) //判断i是否为偶数

continue; //通过continue结束本次循环

System.out.println(i);

}

**Java的数组**

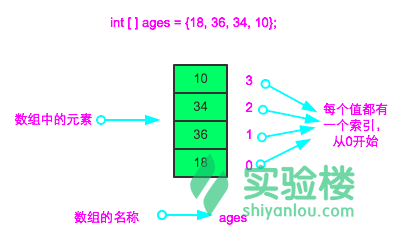
**实验介绍**

　　本实验将学习一维和多维数组的相关知识。

**一、数组的定义与初始化**

　　有些时候，我们需要对很多相同数据类型（java中的数组元素需要相同数据类型）的数进行集中或者统一处理的时候，我们可以利用数组来帮助我们对数据进行管理。

　　数组就是相同数据类型的元素按一定顺序排列的集合,它是有顺序的，它的顺序就是初始化的顺序（双向链表）。我们可以把它看成一个大的盒子，里面按顺序存放了多个数据类型相同的数据。



　　比如定义一个int型数组ages存储四个人的年龄，数组中的元素都可以通过下标来访问，下标从 0 开始。例如，可以通过 ages[0] 获取数组中的第一个元素 18 ，ages[2] 就可以取到第三个元素 34。

　　我们要使用数组，首先我们得对数组进行声明。

语法：

数据类型[ ] 数组名; //或者: 数据类型 数组名[ ];

这样写都是可以的—int 和[]之间有空格；[]之间有空格等：

int [ ] arrInt=new int [] {1,2,3,4};

其中，数组名可以是任意合法的变量名，如：

int ages[]; //存放年龄的数组，类型为整型

char symbol[]; //存放符号的数组，类型为字符型

String [] name; //存放名称的数组，类型为字符串型

　　当我们声明了数组后，需要为数组分配空间，也就是定义多大的数组。

语法：

数组名 = new 数据类型 [ 数组长度 ];

　　数组长度就是数组最多可存放元素的个数。我们也可以在数组声明的时候初始化数组，或者在声明时就为它分配好空间，这样我们就可以不用再为数组分配空间。

语法：

int [] ages = {12,18,9,33,45,60}; //声明并初始化了一个数组，它有6个元素

char [] symbol = new char[10] //声明并分配了一个长度为10的char型数组

分配空间后就可以向数组中放数据了，数组中元素都是通过下标来访问的。

没有显示初始化的数组元素会被隐式初始化，值类型初始化为0等，引用类型初始化为null因为数组是引用类型，所以他的元素值会被自动初始化.

int[] arr=new int[3];  
 System.out.println(arr[0]);

int[] arrInt2;  
arrInt2=new int[3];  
arrInt2[0]=12;  
System.out.println(arrInt2[1]); //输出0  
String[] strArr;  
strArr=new String[2];  
System.out.println(strArr[1]); //输出null

**二、一维数组**

　　数组可以分为一维数组和多维数组。一维数组是由数字组成单纯的线性结构的数组，简单来理解就是一串数。在上面的例子中都是一维数组。

　　一维数组我们可以根据简单地循环进行遍历操作。

比如我们想要在控制台上输出我们上面定义的数组ages。

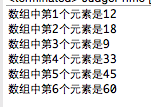
int [] ages = {12, 18, 9, 33, 45, 60};

for(int i = 0; i < ages.length; i++){ //ages.length是获取数组的长度

System.out.println("数组中第"+(i+1)+"个元素是 "+ages[i]); //数组下标是从零开始，一定要注意

}

运行结果：



注意：

1. 数组下标从 0 开始。所以数组的下标范围是0 至 数组长度-1。
2. 数组不能越界访问，否则会报错。

　　 for 语句在数组内可以使用特殊简化版本，在遍历数组、集合时，foreach 更简单便捷。从英文字面意思理解 foreach 也就是“ for 每一个”的意思。

语法：

for(元素类型 元素变量:遍历对象){

执行的代码

}

例如上面的例子：

public class JudgePrime {

public static void main(String[] args){

int [] ages = {12, 18, 9, 33, 45, 60};

int i = 1;

for(int age:ages){

System.out.println("数组中第"+i+"个元素是"+age);

i++;

}

}

}

**三、二维数组**

　　在我们的编程生涯中除了一维数组之外我们还会遇到许多多维数组的应用，其中二维数组是最常见的。

　　二维数组可以看成是一间有座位的教室，座位我们一般用第几排的第几个进行定位，每一个座位都有一个行和一个列的属性，一排的座位相当于一个一维数组，所以我们可以将二维数组简单的理解为是一种“特殊”的一维数组，它的每个数组空间中保存的是一个一维数组。

　　二维数组也需要声明和分配空间。

语法：

数据类型 [][] 数组名 = new 数据类型[行的个数][列的个数];

//或者

数据类型 [][] 数组名;

数组名 = new 数据类型[行的个数][列的个数];

//也可以

数据类型 [][] 数组名 = {

{第一行值1,第一行值2,...}

{第二行值1,第二行值2,...}

...

}

//二维数组的赋值和访问，跟一维数组类似，可以通过下标来逐个赋值和访问，注意索引从 0 开始

数组名[行的索引][列的索引] = 值;

例子：

String [][] name = {{"ZhaoYi", "QianEr", "SunSan"},

{"LiSi", "ZhouWu", "WuLiu"}};

for(int i = 0; i < 2; i++){

for (int j = 0; j < 3; j++){

System.out.println(name[i][j] );

}

}

**Java 方法**

**实验介绍**

　　本实验将学习Java方法的定义与语法规则，参数和返回值的相关内容。

**一、方法的定义与语法规则**

　　Java中的方法，我们可以将其看成一个功能的集合，他们是为了解决特定问题的代码组合。比如司机会开车，老师会教书一样，不同的方法可以实现不同的功能。

方法的定义语法：

访问修饰符 返回值类型 方法名(参数列表){

方法体

}

在上面的语法说明中：

1. 访问修饰符，代表方法允许被访问的权限范围， 可以是 public、protected、private 甚至可以省略 ，其中 public 表示该方法可以被其他任何代码调用，其他几种修饰符的使用我们会在后面章节中详细讲解。
2. 返回值类型是方法返回值的类型，如果方法不返回任何值，则返回值类型指定为 void (代表无类型)（php没有返回值语句return时自动返回null）；如果方法具有返回值，则需要指定返回值的类型，并且在方法体中使用 return 语句返回值。
3. 方法名是方法的名字，必须使用合法的标识符。
4. 参数列表是传递给方法的参数列表，参数可以有多个，多个参数间以逗号隔开，每个参数由参数类型和参数名组成，以空格隔开。

根据方法是否带参、是否带返回值，可将方法分为四类：

* 无参无返回值方法
* 无参带返回值方法
* 带参无返回值方法
* 带参带返回值方法

例如：

public class PrintScore {

public static void main(String[] args) {

// 创建对象，对象名为a

printScore a = new printScore();

int rSum; //设置一个int型变量，用来接收方法的返回值

// 调用方法，传入两门课程的成绩

rSum = a.calcSum(78,99);

System.out.println("总分："+ rSum);

}

/\*

\* 功能：计算两门课程考试成绩的总分并输出总分

\* 定义一个包含两个参数的方法，用来传入两门课程的成绩

\*/

public int calcSum(int a, int b){

int sum= a + b;

return sum;

}

}

　　如上所示：方法名为calcSum，他有两个参数int a和int b，在方法调用的时候必须传入这两个参数。它有一个int 型的返回值，所以在方法里，有一条语句return sum;用来返回sum的值，而在方法调用的时候，我们用一个int 型变量rSum来接收这个返回值。方法calcSum实现了将两门课程的总成绩算出并返回。

注意：

1. 方法体放在一对大括号中，实现特定的操作
2. 方法名主要在调用这个方法时使用，需要注意命名的规范，一般采用第一个单词首字母小写，其它单词首字母大写的形式
3. 调用方法的语法是对象名.方法名（类和对象后面会讲到）

**二、参数与参数列表**

　　　在现实生活中，我们为了完成某件事情之前是需要一些条件的，比如我们买东西肯定是需要钱的。当然，方法也是如此。我们需要方法帮我们完成某项功能的时候，需要根据需求去调整条件。于是就有了参数。

　　　我们可以通过在方法中加入参数列表接收外部传入的数据信息，参数可以是任意的基本类型数据或引用类型数据。

例如：

public void printName(String name){

System.out.println(name);

}

　　上面就是一个方法，方法名为printName，无返回值，参数类型为String，参数名name,它的功能是打印姓名，我们通过参数将姓名传入到这个方法中，利用方法将姓名输出到控制台上。

　　调用带参方法与调用无参方法的语法类似，但在调用时必须传入实际的参数值。语法为对象名.方法名（参数1，参数2...）

　　在这里我们必须区分什么是实参什么是形参。我们先看一个例子。

public class printScore {

public static void main(String[] args) {

// 创建对象，对象名为a

printScore a = new printScore();

double rAve; //用来存放返回的平均成绩

//两课的成绩，是我们实际的参数

int sub1 = 78;

int sub2 = 99;

// 调用方法，传入两门课程的成绩

rAve = a.calcAve(sub1,sub2);

System.out.println("平均分："+rAve);

}

/\*

\* 功能：计算两门课程考试成绩的平均分并输出平均分

\* 定义一个包含两个参数的方法，用来传入两门课程的成绩

\* 两个参数 a 和 b 为我们的形参，只在方法内有效

\*/

public double calcAve(int a, int b){

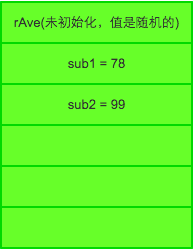
double ave = (a + b) / 2;

return ave;

}

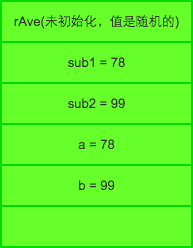
}

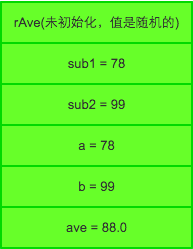
　　我们可以通过内存的图示来简单地说明一下。



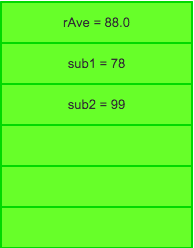
　　上图是方法调用前内存的情况，所有的变量都存放在桟中，由系统统一管理，因为我们在定义变量rAve时未对它进行初始化，所以它的值是随机的，由它所在空间原有的值决定。

　　我们在创建对象时系统也会为对象分配内存空间，这里不详细说明，以后木木会告诉大家。





　　方法调用时，系统会专门为方法开辟一块空间用来存放方法内的变量。此时大家可以看到，sub1和sub2两个实参其实并没有进入方法中，只是将它们两个的值赋值个了形参a和b。



　　方法调用结束后，系统会释放掉方法所产生的空间，回收系统资源。该方法有返回值，并被rAve接收了。

注意：

1. 调用带参方法时，必须保证实参的数量、类型、顺序与形参一一对应
2. 调用方法时，实参不需要指定数据类型
3. 当方法参数有多个时，多个参数间以逗号分隔

**三、返回值**

　　如上面的例子所示，方法是一个实现了条件输入和结果输出的过程，往往我们需要对方法的结果进行进一步的处理，这时候我们就需要方法返回值了。

注意：

1. 如果方法的返回类型为 void ，则方法中不能使用 return 返回值。
2. 方法的返回值最多只能有一个，不能返回多个值。
3. 方法返回值的类型必须兼容，也就是说如果返回值类型为 int ，则不能返回 String 型值
4. 调用带返回值的方法时，由于方法执行后会返回一个结果，因此在调用带返回值方法时一般都会接收其返回值并进行处理

# Java 类与对象

## 实验简介

　　本实验将带领同学们理解面向对象的思想，学习类和对象的概念，以及明确它们之间的关系。

## 一、什么是面向对象

　　同学们，我们终于熬到了面向对象的知识点讲解了。面向对象是学习 Java 必须掌握的知识，也是 Java 里的一个重难点。那什么是面向对象呢？

　　在讲解面向对象之前，我们得解决一个问题，什么是对象？

　　对象，从字面意思来看就是我们面对的物象。于此我们便可以知道，万事万物皆为对象。比如：一台电脑，一辆汽车，一部手机等等都是对象。

　　面向对象，从字面意思来看就是我们人面对着一个对象。其实就是指我们从这个对象的整体出发去看它，它由哪些部件组成，它可以做到哪些事情。

　　比如我们想要买一部手机，我们想要内存大一点的，最新款的，CPU 运算快一点的，能实现发短信和打电话功能的手机。那么这部手机是不是对象呢？它不是。当我们买了一部 iPhone 6 后，它满足我们上面的所有信息。于是我们拿在手上的这部 iphone 6 就是我们的对象。于是我们知道，对象一定是一个具体的，确定的物体。

　而这部手机它的样式，颜色，大小，产地，编号等等，便是这部手机的“属性”，这部手机可以打电话、发短信，便是它的“行为”。

　　面向对象的思想，体现的是人所关注对象的信息聚集在了一个具体的物体上。人们就是通过对象的属性和行为来了解对象。

## 二、类

　　对于一个具体的对象而言，比如上面我们说的那一部 iphone 6，世上还有许多跟这部手机有着同样属性或行为的对象，我们为了方便将它们归类起来，提取出他们相同的属性和行为，而我们把归类起来的这个抽象的概念，称之为类。

　　比如每个人就是一个对象，小张是一个对象，小明是一个对象。而每个人虽然不同，但却有许多相同的属性和行为，于是我们可以他们抽象出来，变成一个类，比如人类。

　　类是封装对象的属性（严格说来是字段或成员变）和行为的载体，反过来说具有相同属性和行为的一类实体被称为类。

　　由此我们可以总结出类的特点：

　　1. 类是相同或相似对象的一种抽象，是对象的一种类型

　　2. 类是具有相同属性和方法（行为）的对象的集合

　　我们在上面反复强调对象的属性和行为，什么是对象的属性呢？什么又是对象的行为呢？

　　属性是对象具有的特征。每个对象的每个属性都拥有特定值。我们上面讲过对象是一个具体并且确定的事物，正是对象属性的值来区分不同的对象，比如我们可以通过一个人的外貌特征区分他。

那什么是对象的行为呢?在计算机中我们通过方法去实现对象的行为，而对象的方法便是对象所具有的操作，比如人会走路、会哭泣、会学习等等都是人的行为，也就是人的方法。

类和对象之间有什么关系吗？在上面的讲解中大家应该有些了解了。类就是对象的抽象，对象就是类的具体。比如手机是一个抽象的概念，它代表着类。而一部 iphone 6 便是手机具象化处理的实体，也就是一个对象。

　　说了那么多，那我们如何在计算机中定义一个类，如何实现一个类呢？ 我们以前说过，Java 是面向对象的语言，而他的体现就在于Java程序都以类 class 为组织单元。而一个类是对象的抽象，所以类由属性和方法两部分组成。

　　定义一个类，主要有三个步骤：

　　1、定义类名，用于区分不同的类。如下代码中 public class 后面跟的就是类名。class是我们声明类的关键字，类名后面跟上大括号，大括号里面就是我们类的一些信息。public我们会在后面详细讲解，大家先记着就行了。

public class 类名{

//定义属性部分（成员变量）

属性1的类型 属性1;

属性2的类型 属性2;

...

//定义方法部分

方法1;

方法2;

...

}

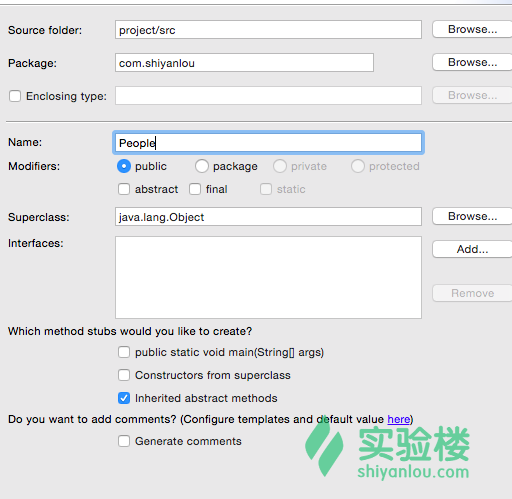
　　2、编写类的属性。对象有什么，我们通过属性来表示。属性的定义我们是写在类名后面的大括号里，在定义属性时，要明确属性的类型。在一个类当中可以写一个或多个属性。当然也可以不定义属性。

　　3、编写类的方法。方法也是写在大括号里面。我们可以定义一个方法或多个方法，当然也可以不定义方法。

接下来我们就来定义一个类吧（其实在以前，我们已经讲过了）。

1、 打开 Eclipse，新建一个 Java 项目，取名为 project。

2、在左侧空白地方点击右键新建一个Class，这便是我们的类了。填上包名（ package ）和我们的类名，至于其他的选项，我们会在后续课程介绍。



3、至此我的类就创建好了，接下来就是对类填写属性和方法了。同学们想一想人这个类有什么属性和方法呢？（当然会很多很多，有些时候我们只关注我们需要的信息就可以了）。



## 三、对象

　　在前面我们已经理解了对象的含义，那么我们该如何将我们定义的类转化为对象呢？我们又该如何使用我们的对象呢？

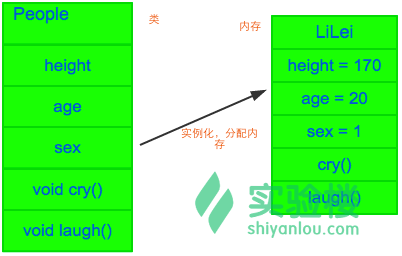
　　我们要使用一个对象，首先第一步便是创建一个对象。创建对象的语法如下：

类名 对象名 = new 类名(); //不像PHP，实例化对象必须带括号，即Hello h=new Hello;是不行的，因为实例化调用的是构造方法

比如对People这个类，我想实例化LiLei这个人。LiLie 的数据类型便是 People 这个类型。（类可以看成是我们自己定义的数据类型）

People LiLei = new People();

　　我们在上一讲中介绍形参和实参的时候画了内存图，定义类的时候是不会为类开辟内存空间的，但是一旦创建了对象（使用new关键字），系统就会在内存中为对象开辟一块空间，用来存放对象的属性值和方法。



创建对象后，我们就要使用对象了，使用对象无非就是对属性和方法进行操作和调用。语法如下

//引用对象属性

对象名.属性

//引用对象方法

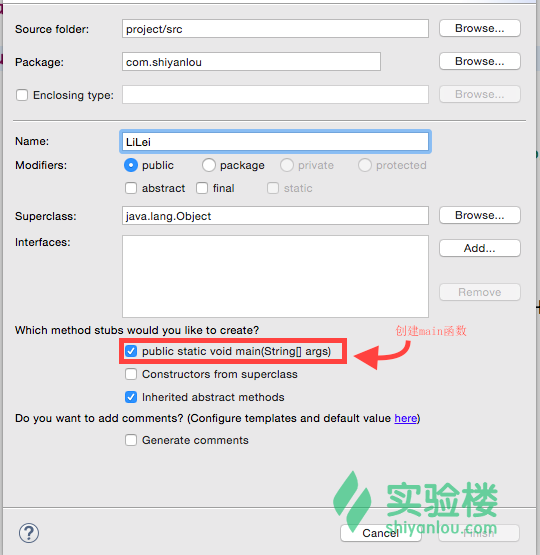
对象名.方法

例如，我们对LiLei的身高赋值，并调用哭这个方法

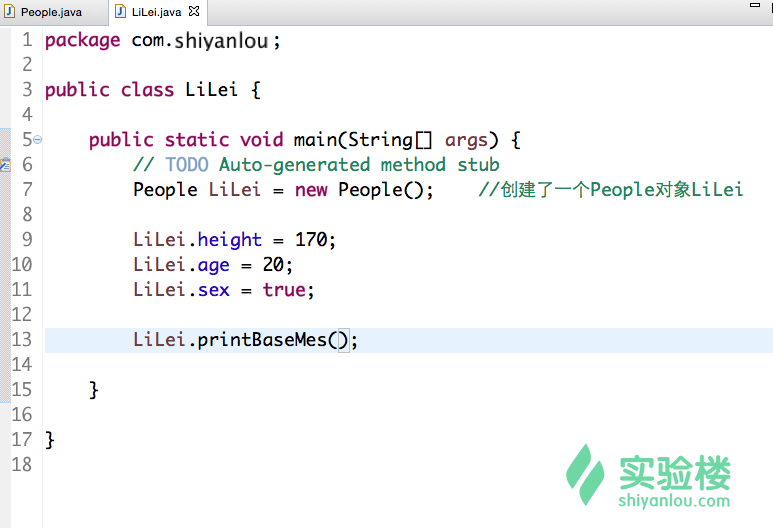
LiLei.height = 170;

LiLei.cry();

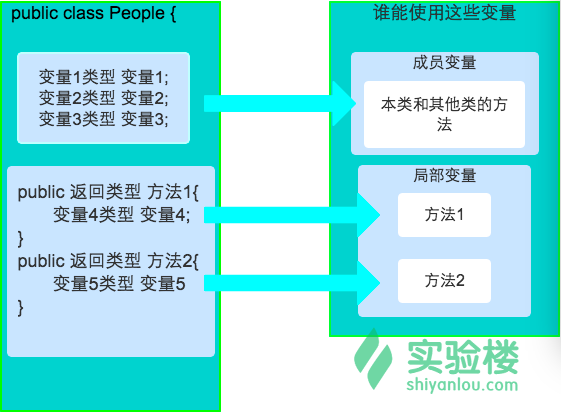
　　接下来我们利用上面的代码来具体操作一下吧。我们在包下点击右键创建一个 Class，取上名字，并选择创建 main 函数。



　　接下来就是在主函数中创建对象，并进行相关的操作了。



　　刚刚我们引入了成员变量这个概念，那什么是成员变量呢？成员变量就是指的对象的属性，是在类中定义，来描述对象的特性。还有一种变量叫局部变量，它是由类的方法定义，在方法中临时保存数据。



　　在使用是注意，我们的成员变量可以被本类的所有方法所使用，同时可以被与本类有关的其他类所使用。而我们的局部变量只能在当前的方法中使用。

　　在这里我们要讲到一个关于作用域的知识了。作用域可以简单地理解为变量的生存期或者作用范围，也就是变量从定义开始到什么时候消亡。

　　1. 局部变量的作用域仅限于定义它的方法内。而成员变量的作用域在整个类内部都是可见的。

　　2. 同时在相同的方法中，不能有同名的局部变量；在不同的方法中，可以有同名的局部变量。

3. 成员变量和局部变量同名时，局部变量具有更高的优先级。

大家可以编写代码验证一下。

## 四、构造方法

　　在面向对象中有一个非常重要的知识点，就是构造方法。我们只前在创建一个对象的时候类名 对象名 = new 类名();通过new关键字将类实例化成对象，而new后面跟的并不是我们的类名，而是与类名相同的一个方法，这个方法便是构造方法。于是可以知道new + 构造方法可以创建一个新的对象，创建对象时实际调用的时候构造方法来创建的，而构造方法是定义在 Java 中的用来初始化对象的方法。

　　1、 构造方法的名称与类名相同，且没有返回值。它的语法格式如下：

//与类同名，可以指定参数，没有返回值

public 构造方法名(){

//不需要指定是否有返回值，否则会被当做普通的方法

public Main Main(){ //这只是定义了一个普通的方法，不会被当做构造方法来看待  
 System.out.println("create the Main");  
 return null;  
}

//初始化代码

}

例如：

package com.shiyanlou;

public class People {

//属性（成员变量）有什么

double height; //身高

int age; //年龄

boolean sex; //性别，true为男性，false为女性

//构造函数，初始化了所有属性

public People(double h, int a, boolean s){

height = h;

age = a;

sex = s;

}

}

//创建对象，调用我们自己定义的有参构造方法

People XiaoMing = new People(168, 21, false);

　　2、如果我们在定义类的时候没有写构造方法，系统会默认给我们生成一个无参构造方法，不过这个构造方法什么也不会做。

　　3、当有指定的构造方法时，系统都不会再为我们添加无参构造方法了。

　　4、构造方法的重载：方法名相同，但参数不同的多个方法，调用时会自动根据不同的参数选择相应的方法。

PHP就不支持方法的重载

# Java 封装

## 实验介绍

　　本实验将学习 Java 面向对象的三大特征之一的封装。

## 一、封装的定义

　　面向对象有三大特性：封装、继承、多态。今天我们就来学习封装。

　　封装就是将类的信息隐藏在类的内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问。那这样的好处是什么呢？

　　　1. 只能通过规定的方法访问数据

　　　2. 隐藏类的实例细节，方便修改和实现。

　　我们在开汽车的时候，只用去关注如何开车，我们并不在意车子是如何实现的，这就是封装。

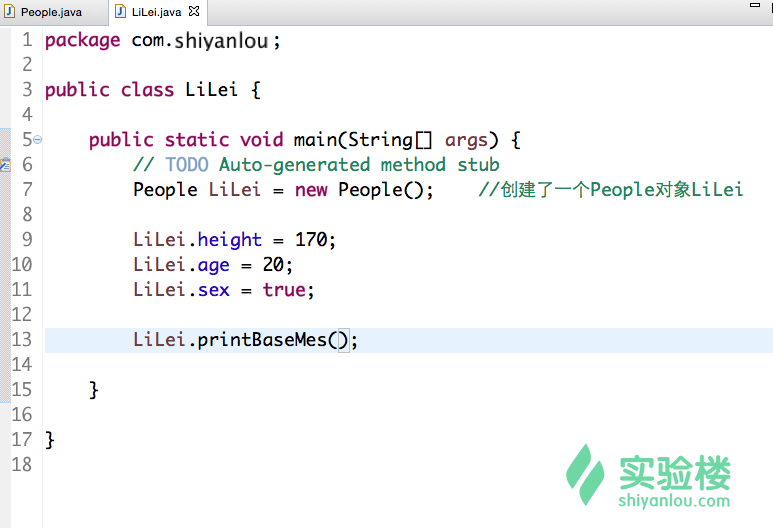
如何去实现类的封装呢？

1. 修改属性的可见性，在属性的前面添加修饰符(private 后面会讲)
2. 创建 getter/setter 方法，用来读写属性
3. 在 getter/setter 方法里加入属性的控制语句，例如我们可以加一个判断语句，对于非法输入给予否定。



　　大家看一下，这是我们上一节课的例子。由于我们没有在属性前面添加任何修饰符，我们通过对象就可以直接对属性值进行修改，没有体现封装的特性。这在许多程序设计中都是不安全的，所以我们需要利用封装，来改进我们的代码。





接下来，我们就改改上面的代码吧。首先我们的类里要将属性前添加private修饰符。然后定义getter和setter方法。

package com.shiyanlou;

public class People {

//属性（成员变量）有什么，前面添加了访问修饰符private

//变成了私有属性，必须通过方法调用

private double height; //身高

//属性已经封装好了，如果用户需要调用属性

//必须用getter和setter方法进行调用

//getter和setter方法需要程序员自己定义

public double getHeight(){

//getter 方法命名是get关键字加属性名（属性名首字母大写）

//getter 方法一般是为了得到属性值

return height;

}

//同理设置我们的setter方法

//setter 方法命名是set关键字加属性名（首字母大写）

//setter 方法一般是给属性值赋值，所以有一个参数

public void setHeight(double newHeight){

height = newHeight;

}

}

然后在我们的 main 函数里的对象，不能再直接调用属性了，只能通过getter和setter方法进行调用。

package com.shiyanlou;

public class LiLei {

public static void main(String[] args) {

// TODO Auto-generated method stub

People LiLei = new People(); //创建了一个People对象LiLei

//利用setter方法为属性赋值

LiLei.setHeight(170.0);

//利用getter方法取属性值

System.out.println("LiLei的身高是"+LiLei.getHeight());

}

}

## 二、包

　　我有时候在封装的时候会遇到这样的问题，就是我们的类名可能是重复的。这时候我们就可以用包的概念来解决我们的问题。

　　包的作用就是管理Java文件，解决同名文件冲突。这和我们生活当中的管理相类似。我们的衣柜是不是有不同的隔断和抽屉，我们将衣服分门别类地放好，是不是有利于我们管理。

　　我们在以前就学习过了如何定义一个包，我们使用package关键字，加上我们的包名。

定义包语法：

package 包名

//注意：必须放在源程序的第一行，包名可用"."号隔开

例如：

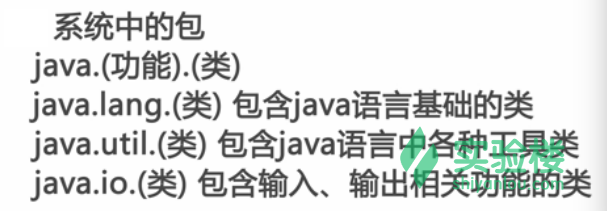
//我们在定义文件夹的时候利用"/"来区分层次

//包中我们用"."来分层

package com.shiyanlou.Java

　　比如我们在游戏里会用到音乐类，在电影里我们也会用到音乐类，但是他们的类又不相同，我们就可以利用包进行区分。

　　不仅是我们这样利用包名来区分类，系统也是这样做的。



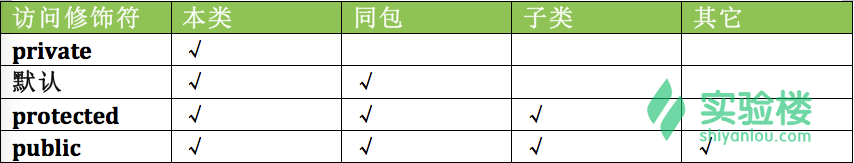
　　那我们怎么在不同包中使用另一个文件中的类呢？这时候就需要用到import关键字。比如我们要导入实验楼下People这个类。import com.shiyanlou.Peole，同时如果import com.shiyanlou.\*这是将包下的所有文件都导入进来。

　　这里要注意一点，包的命名规范是全小写字母拼写。

## 三、Java 中的访问修饰符

　　我们在前面的代码中经常用到private和public修饰符，这些修饰符的作用和意义是什么呢？接下来我们就来学习Java 中的访问修饰符。

　　访问修饰符可以用来修饰属性和方法的访问范围。



　　如图所示，代表了不同的访问修饰符的访问范围，比如private修饰的属性或者方法，只能在本类中访问或者使用。默认是什么修饰符都不加，默认在当前类中和同一包下都可以访问和使用。

　　我们可以举一个例子，比如 money，如果我们用private修饰代表着这是私有的，只能我自己可以使用。如果是protected代表着我可以使用，和我有关系的人，比如儿子也可以用。如果是public就代表了所有人都可以使用。

大家在自己的代码上，修改一下访问修饰符，然后在不同的范围里调用一下你修改的属性和方法。（练习永远比理论更重要！）

严苛程度：

private>默认>protected>public

本类-同包-子类-其他

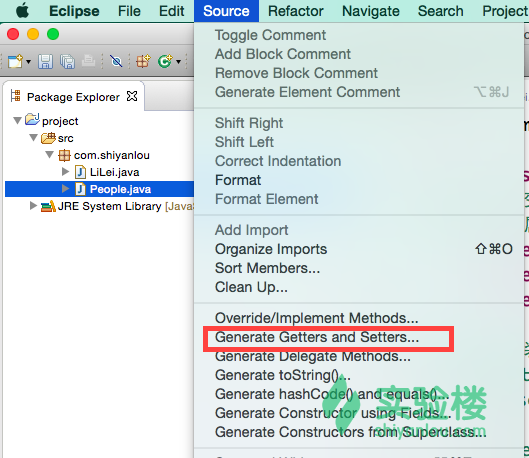
## 四、Java 中的 this 关键字

　　我们在书写代码的过程中常常会用到this关键字，this关键字代表当前对象。使用this.属性操作当前对象的属性，this.方法调用当前对象的方法。

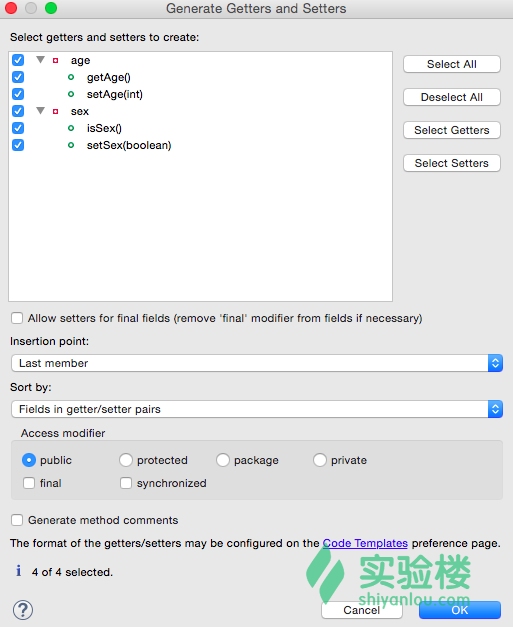
　　当我们封装对象属性的时候，经常会使用this关键字。

　　接下来我们来看看代码中是怎么实现的吧。在上面我们讲了用private修饰的属性，我们利用对象访问的时候，必须定义getter和setter方法，其实在Eclipse中可以自动帮我们定义。

1、首先在菜单栏里点击Source也可以在左侧右击选择Source



2、选择Grenerate Getters and Setters



3、系统会自动帮我们检测需要创建getter和setter方法的属性，我们根据需要勾选就行了。

　　当系统帮我们创建好了getter和setter方法后，我们会发现系统创建的方法中参数名和我们的属性名一样。为了区分参数名和我们的属性名，我们在属性名前面添加了this关键字。

public void setAge(int age) {

this.age = age;

}

此时就代表我们将一个参数的值赋给我们当前对象的属性。同理this关键字可以调用当前对象的方法，同学们自行验证一下吧。

static修饰的属性和方法属于类，因此对象可以访问，但是static修饰的方法不可以访问对象的方法，因为这时对象可能不存在。

## 五、Java 中的内部类

　　从字面意思我们就可以知道，内部类（ Inner Class ）就是定义在另外一个类里面的类，而包含内部类的类被称为外部类。

内部类的主要作用如下：

1. 内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一个包中的其他类访问该类
2. 内部类的方法可以直接访问外部类的所有数据，包括私有的数据
3. 内部类所实现的功能使用外部类同样可以实现，只是有时使用内部类更方便

我们通过代码来详细学习一下内部类吧！

Hello类文件：

private static SubHello \_sh=null;  
public SubHello getSubClass(){  
 if(null==\_sh){  
 \_sh=new SubHello();  
 }  
  
 return \_sh;  
}  
  
  
public class SubHello{  
 public SubHello(){  
 System.out.println("SubHello class is inside of Hello class");  
 }  
}

Main文件：

Hello.SubHello sh=h.getSubClass(); //获得内部类  
System.out.println(sh);

也可以不定义访问变量直接使用外部类来实例化内部类：

Hello.SubHello sh2=h.new SubHello();

#### 成员内部类：

package com.shiyanlou;

//外部类People

public class People {

private String name = "LiLei"; //外部类的私有属性

//内部类Student

public class Student {

String ID = "20151234"; //内部类的成员属性

//内部类的方法

public void stuInfo(){

System.out.println("访问外部类中的name：" + name);

System.out.println("访问内部类中的ID：" + ID);

}

}

//测试成员内部类

public static void main(String[] args) {

People a = new People(); //创建外部类对象，对象名为a

Student b = a.new Student(); //使用外部类对象创建内部类对象，对象名为b

b.stuInfo(); //调用内部对象的suInfo方法

}

}

由此，我们可以知道，成员内部类的使用方法：

1. Student 类相当于 People 类的一个成员变量，所以 Student 类可以使用任意访问修饰符
2. Student 类在 People 类里，所以访问范围在类里的所有方法均可以访问 People 的属性（即内部类里可以直接访问外部类的方法和属性，反之不行）
3. 定义成员内部类（要使用外部类的对象来实例化）后，必须使用外部类对象来创建内部类对象，即 内部类 对象名 = 外部类对象.new 内部类();
4. 如果外部类和内部类具有相同的成员变量或方法，内部类默认访问自己的成员变量或方法，如果要访问外部类的成员变量

#### 静态内部类：

package com.shiyanlou;

//外部类People

public class People {

private String name = "LiLei"; //外部类的私有属性

/\*外部类的静态变量。

Java 中被 static 修饰的成员称为静态成员或类成员。它属于整个类所有，而不是某个对象所有，即被类的所有对象所共享。静态成员可以使用类名直接访问，也可以使用对象名进行访问，但是在类内部static修饰的方法则不能调用非static修饰的属性或者方法。

/\*

static String ID = "510xxx199X0724XXXX";

//静态内部类Student

public static class Student {

String ID = "20151234"; //内部类的成员属性

//内部类的方法

public void stuInfo(){

System.out.println("访问外部类中的ID：" + People.ID);

System.out.println("访问内部类中的ID：" + ID);

}

}

//测试成员内部类

public static void main(String[] args) {

Student b = new Student(); //直接创建内部类对象，对象名为b

b.stuInfo(); //调用内部对象的suInfo方法

}

}

静态内部类是 static 修饰的内部类，这种内部类的特点是：

1. 静态内部类不能直接访问外部类的非静态成员，但可以通过 new 外部类().成员 的方式访问
2. 如果外部类的静态成员与内部类的成员名称相同，可通过类名.静态成员访问外部类的静态成员；如果外部类的静态成员与内部类的成员名称不相同，则可通过成员名直接调用外部类的静态成员
3. 创建静态内部类的对象时，不需要外部类的对象，可以直接创建 内部类 对象名= new 内部类();

public static class SubHello2{  
 SubHello sh=new Hello().getSubClass(); //内部静态类访问外部非静态方法需要new 外部类  
 public SubHello2(){  
 test2(); //静态类成员可以直接访问外部类的静态方法  
 }  
}

Hello.SubHello sh2=h.new SubHello(); //成员内部类属于对象，因此需要对象来实例化  
Hello.SubHello2 sh22=new Hello.SubHello2(); //静态类成员属于类，直接new 即可

#### 方法内部类

　　方法内部类就是定义在外部类的方法中的内部类，方法内部类只在该方法的内部可见，即只在该方法内可以使用。

例如：

package com.shiyanlou;

//外部类People

public class People {

//外部类中的方法

public void peopleInfo() {

final String sex = "man"; //外部类方法中的常量

class Student {

String ID = "20151234"; //内部类中的常量

public void print() {

System.out.println("访问外部类的方法中的常量sex：" + sex);

System.out.println("访问内部类中的变量ID:" + ID);

}

}

Student a = new Student(); //创建方法内部类的对象

a.print();//调用内部类的方法

}

//测试方法内部类

public static void main(String[] args) {

People b = new People(); //创建外部类的对象

b.peopleInfo(); //调用外部类的方法

}

}

运行结果：

运行结果

方法内部类不能在方法外部访问，因此基本就是在方法内部定义和使用。

public void helloInfo(){  
 class cc{  
 public String name="helloInfo";  
 public int age=12;  
  
 public void printInfo() {  
 System.out.println(String.format("name: %s, age: %d", name, age));  
 }  
 }  
  
 cc c=new cc();  
 c.name+=" is a good name";  
 c.printInfo();  
}

**Java 继承**

**实验介绍**

　　本实验将介绍继承，以及父类和子类之间初始化顺序等相关知识。

**一、继承的定义**

　　继承可以看成是类与类之间的衍生关系。比如狗类是动物类，牧羊犬类又是狗类。于是我们可以说狗类继承了动物类，而牧羊犬类就继承了狗类。于是狗类就是动物类的子类（或派生类），动物类就是狗类的父类（或基类）。

在 Java 中一个类只有一个父类，所以 Java 中的继承是单继承。就像我们每个人只有一个亲生父亲，而我们如果有一个有钱的父亲我们就是富二代；有一个明星父亲，我们就是星二代。继承可以让子类直接拥有父类的所有属性和方法(属性和方法的修饰符是 private 就不行，因为是私有的嘛)。而继承的好处便是可以实现代码的复用，是我们程序猿的福利啊！

我们创建一个类需要它来继承某一个类的语法：

class 子类 extends 父类

例如我们定义了一个 Animal 类，我们再创建一个 Dog 类，我们需要它继承 Animal 类

class Dog extends Animal {

...

}

接下来我们就来练习一下吧！

我们先创建一个父类 Animal

package com.shiyanlou;

public class Animal {

public int legNum; //动物四肢的数量

//类方法

public void bark() {

System.out.println("动物叫！");

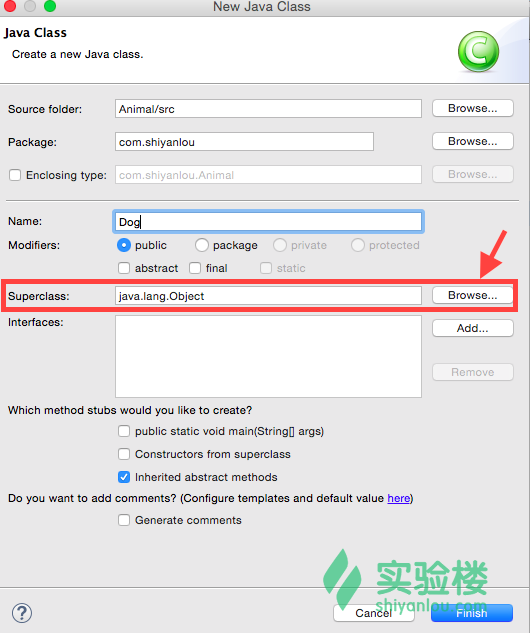
}

}

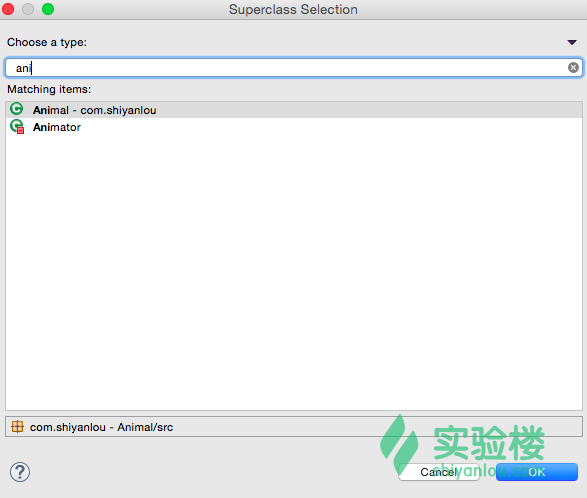
接下来创建一个子类，我们可以自己输入代码，也可以利用 Eclipse 帮我们自动建立继承关系。

1、点击创建 class 类

2、输入子类名称，点击Superclass后面的Browse...寻找父类，与之建立继承关系



3、在Choose a type:下的输入框内输入要寻找的父类，然后选择，结束。

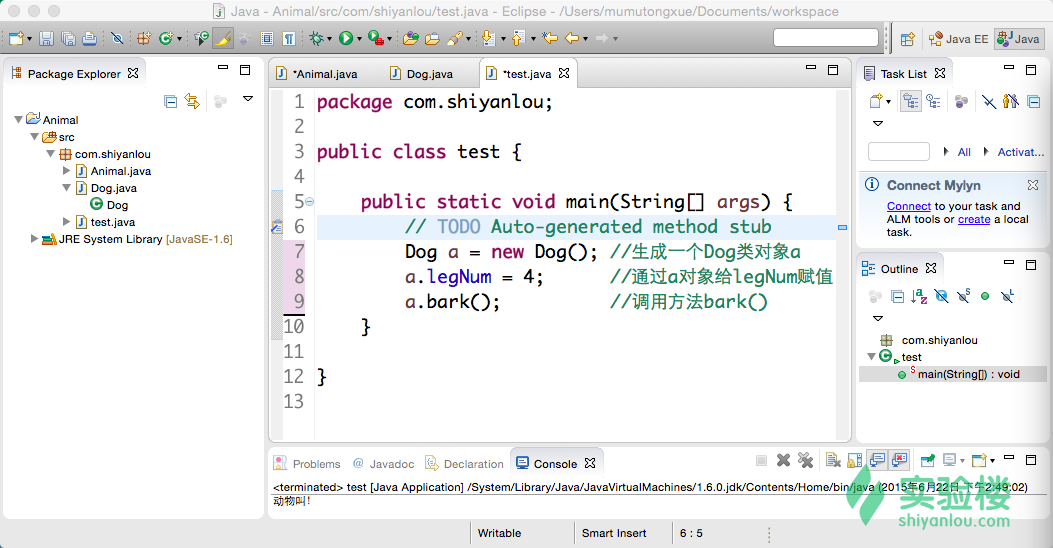


package com.shiyanlou;

public class Dog extends Animal {

}

上面便是系统生成的代码，Dog 类继承了父类 Animal，我们 Dog 类里什么都没有写，其实它继承了父类 Animal，所以 Dog 类拥有 Animal 类的全部方法和属性（除开私有的方法和属性）。我们创建一个测试类测试一下。



**二、方法的重写**

虽然子类可以继承父类的方法，但如果子类对父类的方法不满意，子类想在里面加入适合自己的一些操作时，我们就需要将方法进行重写。并且子类在调用方法中，优先调用子类的方法

比如上一个例子中 Animal 有bark()这个方法代表了动物叫，但是不同的动物有不同的叫法，比如狗是汪汪汪，猫是喵喵喵。

当然在方法重写时我们需要注意，重写的方法一定要与原父类的方法语法保持一致，比如返回值类型，参数类型及个数，和方法名都必须一致。

例如：

package com.shiyanlou;

public class Dog extends Animal {

//重写父类方法

public void bark() {

System.out.println("汪！汪！汪！");

}

}

我们在运行程序后，发现控制台上是不是输出了我们预想的结果呢？

**三、继承的初始化顺序**

我们学习了继承后，知道在一个继承关系中一定会有一个父类和至少一个子类，那么在程序运行的过程中，是为父类进行初始化，还是先调用的子类进行初始化的呢？

继承的初始化顺序是先初始化父类再初始化子类。

　　我们根据代码来验证一下。

package com.shiyanlou;

public class Animal {

public int legNum = 4; // 动物四肢的数量

//Animal的构造方法

public Animal(){

System.out.println("动物有" + legNum + "腿");

System.out.println("父类的构造方法被调用！");

}

}

package com.shiyanlou;

public class Dog extends Animal {

//Dog类的构造方法

public Dog(){

System.out.println("子类的构造方法被调用！");

}

}

package com.shiyanlou;

public class test {

public static void main(String[] args) {

// TODO Auto-generated method stub

Dog a = new Dog(); //生成一个Dog类对象a

}

}

最后输出结果：

输出结果

由此可知，系统先创建了父类对象，再创建了子类对象，先初始化了属性，再调用了构造函数。

最后再讲一个小知识。

final关键字可以修饰类、方法、属性和变量

1. final 修饰类，则该类不允许被继承
2. final 修饰方法，则该方法不允许被覆盖（重写）
3. final 修饰属性：则该类的属性不会进行隐式的初始化（类的初始化属性必须有值）或在构造方法中赋值（但只能选其一）
4. final 修饰变量，则该变量的值只能赋一次值，即变为常量

super关键字在子类内部使用，代表父类对象。

1. 访问父类的属性 super.属性名
2. 访问父类的方法 super.bark()

同学们可以自行验证一下。

# Java 多态

## 实验介绍

　　本实验将学习 Java 面向对象的最后一个特性——多态

## 一、多态定义

　多态就是指对象的多种形态。木木将从两个方面为同学们介绍多态。一个是引用的多态；一个是方法的多态。

#### １、引用多态

　　当我们在写面向对象程序的时候，一般父类的引用可以指向本类的对象。

Animal a = new Animal(); //a是父类的引用指向的是本类的对象

Animal b = new Dog(); //b是父类的引用指向的是子类的对象

//注意：我们不能使用一个子类的引用去指向父类的对象

多态中，可以使用父类的引用指向子类的对象，如上面代码里的对象b，那b究竟是代表 Dog 类还是代表Animal类呢？

HelloChild hc=new HelloChild();  
hc.test();  
Hello hcc=new HelloChild(); //Hello 是HelloChild的父类  
hcc.helloInfo();

#### 2、方法多态

一般情况下，我们都是用本类的对象，调用本类的方法。在多态中，创建子类对象时，调用的方法为子类重写的方法或继承的方法。

Hello hcc=new HelloChild();  
hcc.test(); //hcc是Hello的引用，但是调用的是HelloChild的方法  
System.out.println(hcc.getClass()); //输出：class com.test.testcodes.HelloChild

　　我们在上节课在 Dog 类中重写了bark()方法，我们看一看下面的代码

Animal a = new Animal(); //a是父类的引用指向的是本类的对象

Animal b = new Dog(); //b是父类的引用指向的是子类的对象

//这边是引用的多态

a.bark(); //调用的是父类Animal的方法

b.bark(); //调用的是子类Dog重写的方法

//这便是方法的多态

注意：

1. 父类引用指向子类对象时不可以调用子类独有的方法，只能调用继承或重写的方法
2. 引用多态不能使用一个子类的引用去指向父类的对象

## 二、多态引用类型转换

　　在我们使用多态时，经常会用到引用类型转换。引用类型转换有两种。

　　一种引用类型转换是向上类型转换（隐式/自动类型转换），是小类型到大类型的转换（无风险）。另一种是向下类型转换（强制类型转换），是大类型到小类型（有风险）。

　　比如一杯水，我们倒进壶里，这种向上转换是不存在风险的，计算机可以自动帮我们完成，所以向上类型转换又叫做自动类型转换或隐式类型转换。而如果我们想将壶里的水倒进杯子里，就可能会产生溢出，这是有风险的。计算机是不会自动完成这样的操作。但如果程序猿想要完成这样的操作，就只能强制转换。所以向下类型转换又叫做强制类型转换。

例如：

Dog a = new Dog();

Animal b = a; //自动类型提升 向上类型转换（无风险）

/\*

Dog c = b;

//将父类引用转换成子类引用，编译器不允许这样做

\*/

Dog c = (Dog)b; //在父类前加上小括号，里面加上子类类型，实现强制转换

//向下类型转换

/\*

Cat d = (Cat)b;

//虽然编译器不会报错，但运行会出问题，因为b是指向Dog类的，它们是不同类型的对象，这里就存在风险

/\*

## 三、 抽象类

　　抽象类在定义类时，前面会加abstract关键字。那我们什么时候会用到抽象类呢？

1. 在某些情况下，某个父类只是知道其子类应该包含怎样的方法，但无法准确知道这些子类如何实现这些方法。也就是说抽象类是约束子类必须要实现哪些方法，而并不关注方法如何去实现。
2. 从多个具有相同特征的类中抽象出一个抽象类，以这个抽象类作为子类的模板，从而避免了子类设计的随意性。

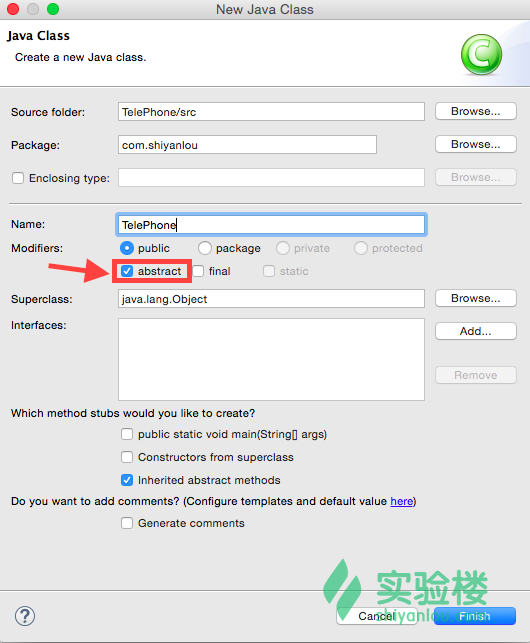
所以由上可知，抽象类是限制规定子类必须实现某些方法，但不关注实现细节。

那抽象类如何用代码实现呢，它的规则如下：

1. 用 abstract 修饰符定义抽象类
2. 用 abstract 修饰符定义抽象方法，只用声明，不需要实现
3. 包含抽象方法的类就是抽象类
4. 抽象类中可以包含普通的方法，也可以没有抽象方法
5. 抽象类的对象不能直接创建，我们通常是定义引用变量指向子类对象。

我们来写一写代码吧

1、先创建一个抽象类



2、 填写需要子类实现的抽象方法

package com.shiyanlou;

//抽象方法

public abstract class TelePhone {

public abstract void call(); //抽象方法,打电话

public abstract void message(); //抽象方法，发短信

}

3、构建子类，并实现抽象方法。

package com.shiyanlou;

public class CellPhone extends TelePhone {

@Override

public void call() {

// TODO Auto-generated method stub

System.out.println("我可以打电话！");

}

@Override

public void message() {

// TODO Auto-generated method stub

System.out.println("我可以发短信！");

}

}