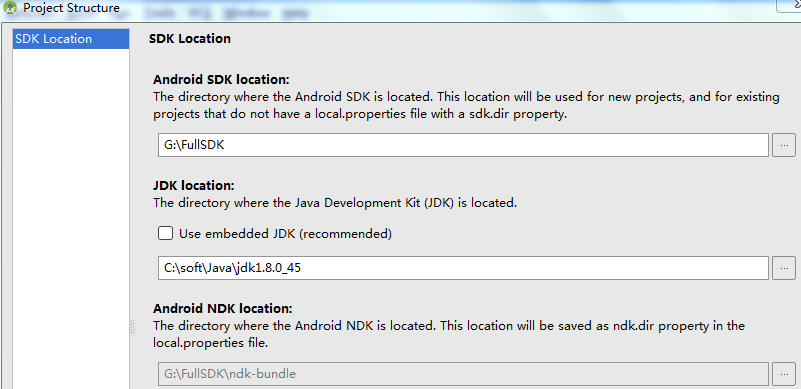
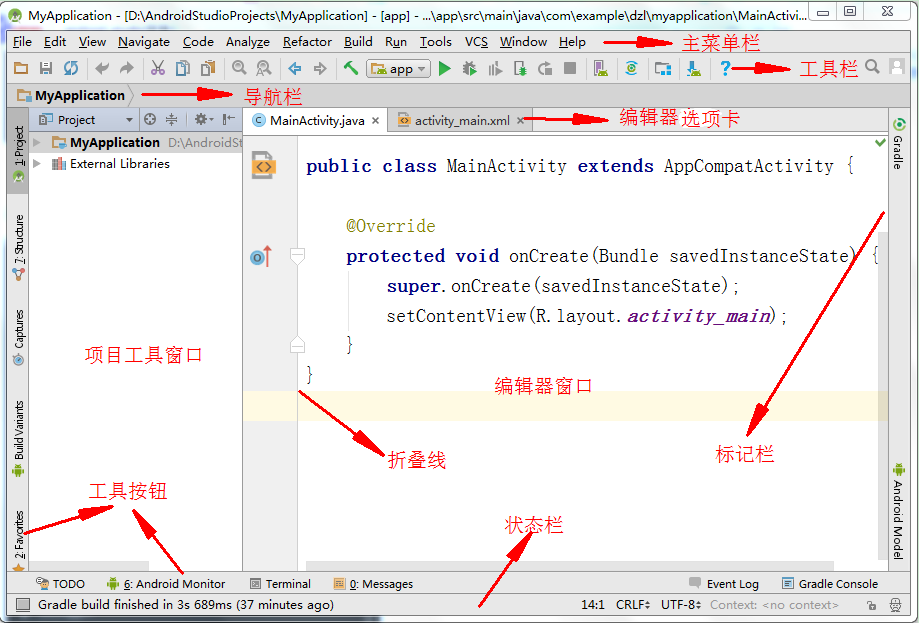
# 第一章：AndroidStudio入门

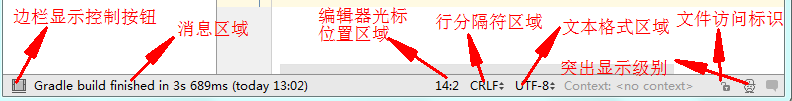
1、安装JDK1.8，设置JAVA\_HOME，现在最新版本的AndroidStudio不会使用这个JDK，需要在AndroidStudio中配置：File🡪Other Settings🡪Default Project Structure,如下图：

2、安装AndroidStudio

# 第二章：在AndroidStudio中遨游

## 认识AndroidStudio窗口



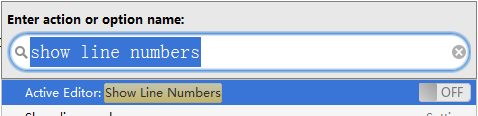
* 编辑器：总是位于中心面板，其它所有的窗口均被称为工具窗口，它们位于编辑器窗口的周围（左侧、下方和右侧）
* 编辑器选项卡：在编辑器选项卡之间导航：Alt + 左右箭头，关于编辑器选项卡的命令在：Window🡪Editor Tabs，在编辑器选项卡上右击可出现相关的上下文菜单。
* 折叠线：折叠代码，也可用于设置断点。
* 标记栏：标记代码，标记自动出现，如有报错的会有红色标记，这些标记不会跟随滚动条滚动，它显示的位置是文件的相对比例的位置，点击可以跳转到标记的代码的位置。
* 工具按钮：通过View🡪Tool Windows可查看所有的工具按钮，有些工具按钮上标有数字，意味着可以按下Alt和对应的数字打开该工具窗口。如Alt + 1打开Project工具窗口
* Window菜单中有保存布局和恢复默认布局的命令
* Project工具窗口：最常用的工具窗口，显示项目文件
* Stucture工具窗口：显示文件元素的层次结构，如果是Java类，则会显示字段、方法和内部类等元素，如果是XML则显示XML的树结构，点击Structure中的任意元素则会跳转到编辑器中的对应位置，这在较大的文件中导航非常有用。
* Favorites工具窗口：用于收藏文件、书签、断点，在编辑器选项卡有右击🡪Add toFavorites/Add All to Favorites即可收藏文件，书签功能则可以让你收藏代码的某一行，定位到代码的某一行，然后按下F11则可以把该放入书签，在该行的左侧会有一个勾。在折叠线上单击则可以增加断点，在折叠线左侧单击也可以，并且会显示红点，该行代码也会有红色背景显示。
* TODO工具窗口：代办事项，TODO本质上就是注释，用于提醒自己或者其它同事还有尚未完成的工作。
* 主菜单栏：通过它几乎可以执行任何操作，它无法隐藏，不要怕命令多，即使是最教练的Android开发者在日常工作中也仅仅会用到这些操作的一小部分而已，并且大多数命令均有快捷键或上下文菜单。
* 工具栏：包含频繁使用的操作。View🡪Toolbar可隐藏
* 导航栏：用于在项目的资源之间进行导航。
* 状态栏：显示相关上下文敏感的反馈。  
  
  + 边栏显示控制按钮：点击可以显示或隐藏窗口左右两边的工具按钮，把鼠标放到上面还可以切换显示工具窗口
  + 提供反馈并显示关于当前正在运行进程的所有信息。当把鼠标移到菜单项或工具栏中的按钮等UI元素时，此区域还会显示提示，在这个区域单击会打开Event日志。
  + 编辑器光标位置区域：显示当前行列位置，点击可输入行进行定位（列可以输入也可不输入）
  + 行分隔符区域：显示了文本文件中使用的回车换行格式。在Windows上，默认值是CRLF，表示回车换行符，LF是Unix和Mac机器上使用的标准格式，也是Git中所采用的。如果在Windows电脑上进行开发，Git通常会在向仓库提交代码时将CRLF转换为LF。
  + 文本格式区域：描述了用于源文件的文本编码。默认值是UTF-8，这旨ASCII的超集且涵盖了大多数西方字符，包括你能够在标准Java或XML文件中找到的所有字符。
  + 文件访问标识符区域：允许你在文件的读/写和只读之间进行切换。
  + 突出显示级别区域：允许你设置想要在代码中看到的突出显示级别。默认设置是Inspections（检查），此设置意味着你要准备好授受严格的检查，因为Inspections Manager会严格识别代码中的语法错误和问题（称为警告）。你会看到Inspections Manager在标记栏中以黄色标记的形式生成的一些警告。如果将级别设置为Syntax，则Inspections Manager会忽略警告。虽然没有Inwspections模式严格，但是仍然会突出显示那些将会阻止代码编译的语法问题。另一个模式为None，这意味着Inspections Manager已经不再关心你的代码了，在些模式下，即使是最严重的语法错误也会被忽略。

## 常用操作

* 双击单词可选择整个单词，在一行中三击，则选中整行。
* 结构化选择：将光标置于任意单词，按下Ctrl + W，选择整个单词，继续按，则选中的区域会扩展至包含任意数量的相邻文本，按Ctrl + Shift + W，则选中区域会缩小
* Undo和Redo：Ctrl + Z和Ctrl + Shift + Z
* 找到最近的文件：AndroidStudio最优秀的特性还包括它会记录你在近期操作过的所有文件：View🡪Recent Files，或者按Ctrl + E，它会列出最近修改过的文件，最多50个，可以通过File🡪Settings🡪Limits🡪Editor🡪Recent Files Limit修改这个限制。
* 遍历最的的导航操作：AndroidStudio还会记住你最近的导航操作，包含光标移动、选项卡切换和文件打开。要遍历你的导航操作历史，按Ctrl + Alt + 左右箭头。
* 复制：
  1. 复制路径：Ctrl + Shift + C，可以复制Project工具窗口以及任意编辑器选项卡中选中的文件或目录的全路径
  2. 复制引用：Ctrl + Alt + Shift + C，可以复制方法、变量或类的逻辑引用。

## 获取帮助

Help菜单中有几个有用的菜单项。

File Action（Ctrl + Shift + A）是在AS中获取帮助 的最常用命令，此命令会打开一个对话框，允许你搜索AS中所有特性，比如输入Show Line Numbers，如下：  
  
点击OFF开关即可打开，在编辑器窗口中即可显示行号，在Settings🡪Editor🡪Appearance中也能看到Show Line Numbers复选框。

Help🡪Android Studio Help是所有AS技术规范文档的来源，这是关于AS最详细的文档。

Help🡪 Keymap Reference是AS所有的快捷键

## 使用键盘导航

键盘或许是AS中导航的最有效方式了。查看Navigate菜单。

Select In： Alt + F1

Class：Ctrl + N，导航到指定的Java类，包含内部类

File：Ctrl + Shift + N，导航到项目中的任意文件，如果正在查找项目中的XML文件，可以采用这种操作方法

Line：Ctrl + G，导航到源文件中的某一行

Related File：Ctrl + Alt + Home，打开相关联的文件，如打开Activity关联的XML布局文件。

Last Edit Location：Ctrl + Shift + Backspace，导航到最后一次编辑的地方。可以连续向后导航。

Type Hierarchy：Ctrl + H，列出Java类的继承树结构图。

Declaration：Ctrl + B，跳转到字段或方法的声明出。

## 查找和替换文本

Find：Ctrl + F 在单独的文件中查找，在标记栏会标记出查找到的位置，点击可直接跳转

Find in Path：Ctrl + Shift + F 在指定的路径下进行搜索，可以使用正则表达式，并使用文件掩码来限定结果范围

Replace：Ctrl + R 它是Find的超集，用于替换文本，更安全的做法应该使用Refector🡪Rename

Replace in Path：Ctrl + Shift + R 是Find inPath的超集，用于替换指定路径下的文本，更安全的做法应该使用Refector🡪Rename

# 第三章 在AndroidStudio中编程

关于代码操作的许多快捷功能都在Code菜单，如果记不住快捷键，则可以使用Code菜单中的命令。

## 3.1代码折叠

Settings🡪Editor🡪General🡪Code Folding可修改代码折叠选项

Expand（展开）：Ctrl + 数字加号，展开光标所处位置的已折叠代码块

Collapse（折叠）：Ctrl + 数字减号，折叠光标处已展开的代码块

Expand All：Ctrl + Shift + 数字加号，展开窗口中的全部代码

Collapse All：Ctrl + Shift + 数字减号，折叠窗口中的全部代码

Toggle Fold：Ctrl + 句号，折叠/展开光标所处位置的代码块

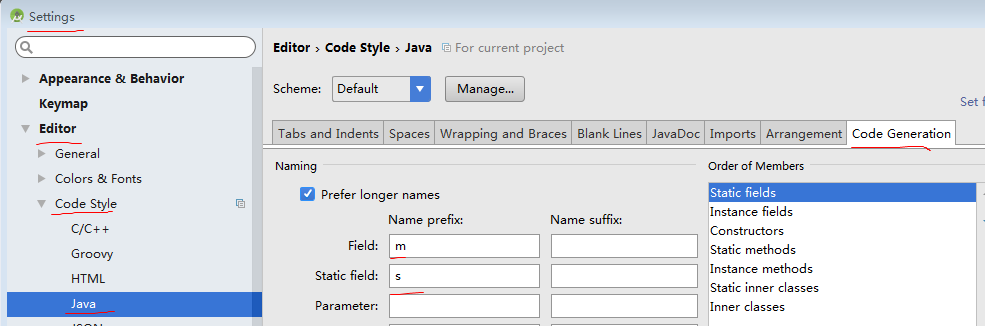
## 3.2 执行代码补全

AS的4种代码补全类型：

1. Default代码补全，无快捷键，会在你开始输入的时候自动出现一个建议列表，按上下箭头选择，回车确认
2. Basic代码补全，Ctrl + 空格，类似于Default代码补全，但还会在建议列表中当前选中条目的旁边显示一个Javadoc窗口，单击这个窗口中的向上箭头可显示详细文档
3. SmartType代码补全，Ctrl + Shift + 空格
4. Cyclic Expand Word代码补全，Alt + /，输入单词前面的1 ~ n个字母后，按此快捷键，提供文档中使用过的单词，向上循环。
5. Cyclic Expand Word代码补全，Alt + Shift + /，输入单词前面的1 ~ n个字母后，按此快捷键，提供文档中使用过的单词，向下循环。

## 3.4使用代码生成

Alt + Insert，生成构造函数、getter、setter、equals()、hashCode()、toString()等等。

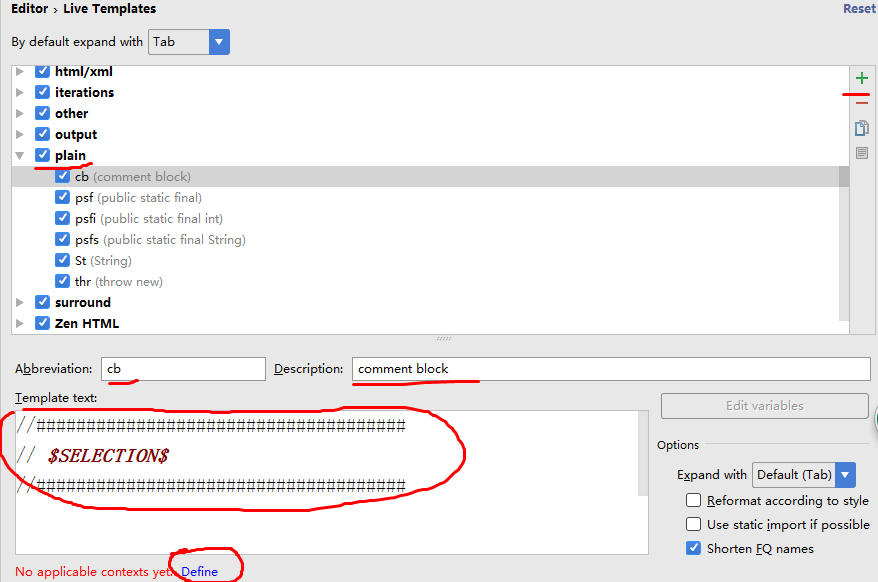
配置忽略成员变量名称的前缀m和s：File🡪Settings🡪Editor🡪Code Style🡪Java🡪Code Generation，如下：  


AS掌握类成员的信息，因而允许在类中定义代理方法并将行为代理到类成员的方法，比如在为类中有一个成员变量为private List<String> names，按下Alt + Insert，选择Delegate Methods，选择names，再选择需要代理的方法即可，如果需要代理多个方法可按住Ctrl键选择多个方法。

## 3.5 插入动态模板

在许多IDE中，生成的代码只是从模板中粘贴过来，并未考虑作用域，AS的模板是作用域敏感的，而且还可以集成变量数据。

创建AS模板：File🡪Settings🡪Editor🡪Live Templates🡪选择一个模板分组，如选择plain分组，然后点击右上角的加号按钮，再选择Live Templates，然后填写Abbreviation、Description和Template text，然后点击窗口底部的Define按钮，这个按钮看起来像一个蓝色的超链接，点击它选择Java并选择所有范围（语句、表达式、声明等，这代表在Java源文件的任何作用域都有效），点击Apply按钮，如下：



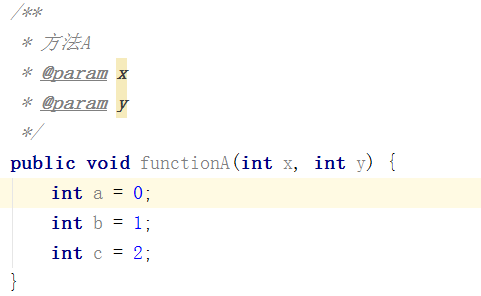
上图模板中的$SELECTION$是一个变量，表示选择的内容，在字段或方法的上方输入注释文字，然后选中这些文字，按Ctrl + Alt + J，选择cb即可自动生成注释。

* Insert Live Templates：Ctrl + J，激活作用域敏感的Live Templates列表，其实直接输入模板名称就会提示列表，如果不小心列表隐藏了，则再按Ctrl + J来显示。
* Surround with Live Templates：Ctrl + Shift + J，激活作用域敏感的Surround with Live Templates列表。在模板中的模板，如果作用域符合，并且模板中有变量$SELECTION$，则可以使用这个快键，而且有$SELECTION$变量的模板使用Ctrl + J是调不出来的。

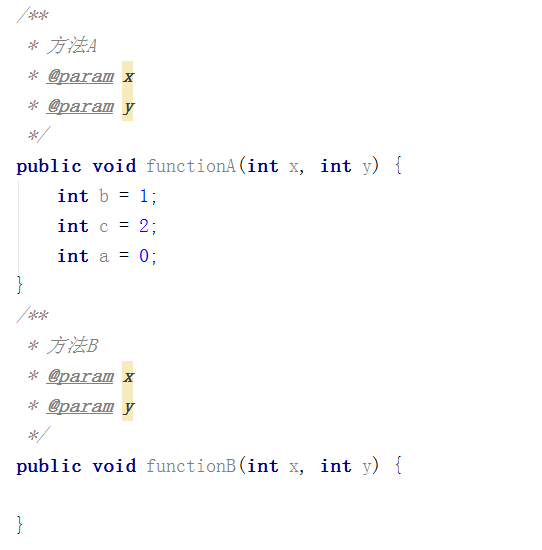
## 3.6 移动代码

Ctrl + Shift + 上下箭头移动，这种移动会限制在作用域范围内，

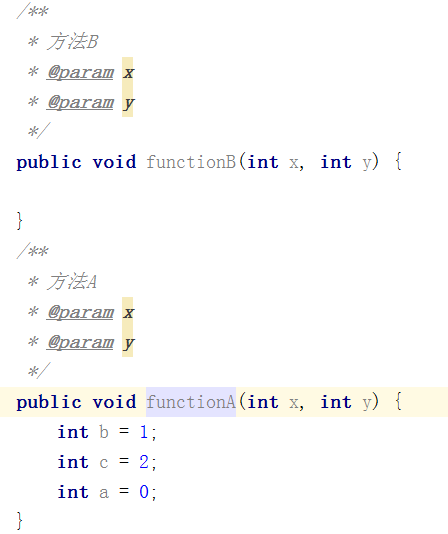
Alt + Shift + 上下箭头移动，这种移动没有作用域的限制，如下：



把光标定位在int a = 0这一行，然后按下Ctrl + Shift + 箭头下，会发现这一行代码可以往下移动两行到int c = 2的后面，再按就不会移动了，因为如果再往下移动的话就超出这行代码的作用域了。如果希望可以一直往下移动，则使用Shift + Alt + 下箭头。

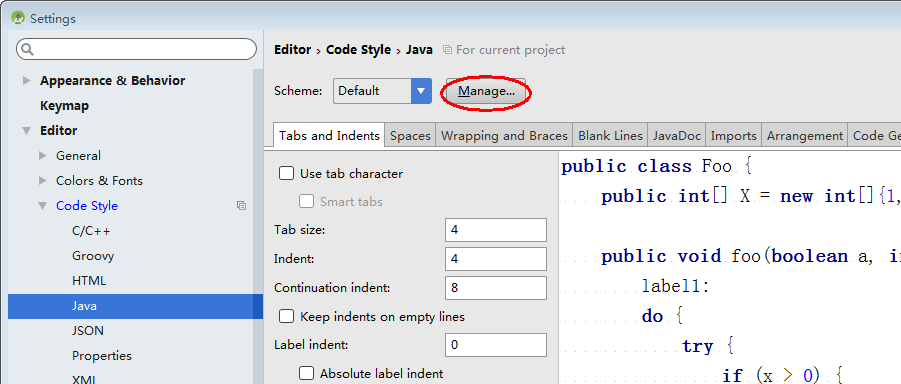


把光标定位在functionA的这一行，然后按下Ctrl + Shift + 箭头下，会发现连同注释一起移到了functionB下，如下：

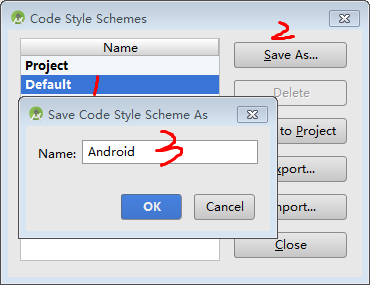


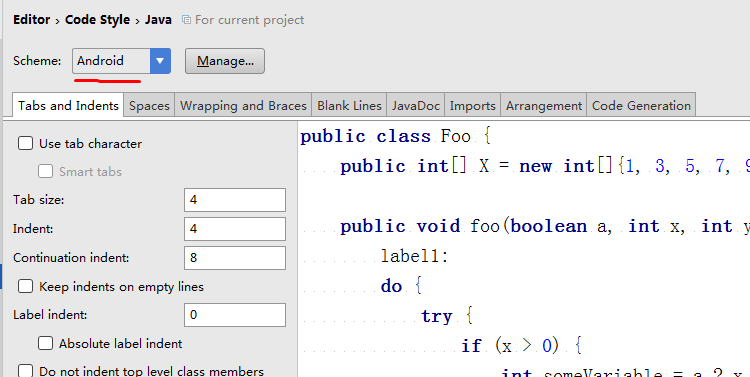
## 3.7 设计代码风格

代码风格约定处于不断演进之中。有关在方法后面应该留多少个空格，或者开始花括号应该与方法签名出现在同一行还是在下面一行，这些内容本来就没有定论。不同的组织倾向于定义其自身的代码风格，但每个程序员的代码风格也各不相同；况且你也可以有一种自己习惯的代码风格。AndroidStudio可以让你设计自己的代码风格，在File🡪Settings-🡪Editor🡪Code Style，Java和XML是我们在Android开发中需要关注的语言。操作如下：



点击“Manage。。。”，弹出如下对话框：





在这个对话框中，我们另存了一个配置，所以可以随意修改了。 当我们按Ctrl + Alt + L 来格式化代码时，上面的代码风格设置会被应用。

### 3.7.1 自动缩进行

快捷键：Ctrl + Alt + I，它会把选中行的代码进行缩进，缩进的规则是在File🡪Settings🡪Editor🡪Code Style🡪Java🡪Tabs and Indents中设置的。

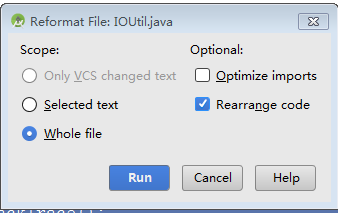
### 3.7.2 重新排版代码顺序

Arrangement管理着代码中元素的顺序。例如，大多数人喜欢将类成员声明置于类的顶部，接着是构造函数，然后是getter和setter，等等。可以在Arrangement选项卡中编辑Arrangement设置：File🡪Settings🡪Editor🡪Code Style🡪Java🡪Arrangement。

当需要重新排版代码顺序时，使用命令：Code🡪Rearrange Code即可。

### 3.7.3 重新格式化代码

这是最强大的Code Style操作，因为它会应用Code Style设置中定义的所有代码风格。快捷键为Ctrl + Alt + L，经本人试验，该快捷键并没有应用Code Style中的所有设置，如果想要应用更多，可以按Ctrl + Shift + Alt + L，这时会弹出一个对话框，可选择应用更多的Code Style，如下：



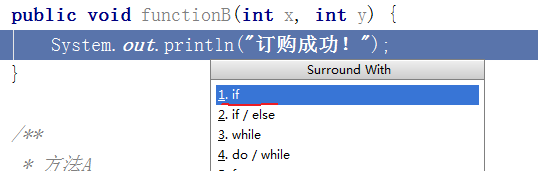
* Selected text：对选择的文本生效
* Whole file：整个文件生效
* Optionze imports：删除没有使用到的导入语句，这个优化可直接按Ctrl + Alt + O
* Rearrange code: 重新排版代码格式，这个优化可点击菜单Code🡪Rearrange code

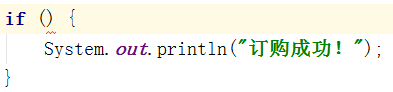
### 3.7.4 Surrond with

Surround With（Ctrl + Alt + T），它是Surround with Live Template（Ctrl + Alt + J）功能的超集。不过，Surround with还包含使用Java块（例如if/else、for、try/catch等，这些不属于模板）来包裹选中的一条或多条语句的选项。使用try/catch块包裹可能会抛出异常的语句是Surround with的最常用的应用场景之一，这或者是Ctrl + Alt + T 使用T的原因。

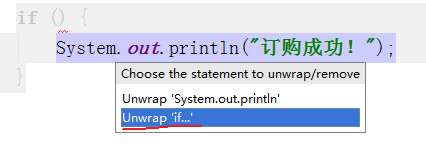
Ctrl + Shift + Delete则是删除使用的代码块，示例如下：

选中需要应用代码块的代码，如果是单行，则可以在单行上单击，然后就可以按下Ctrl + Alt + T了，如下：





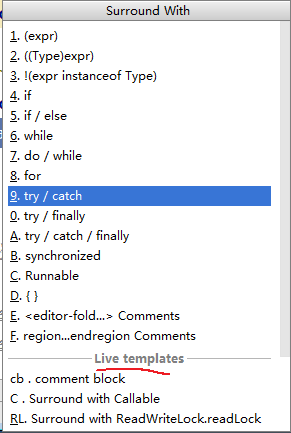
在if块内任意位置单击，然后按下Ctrl + Shift + Delete，如下：



结果if块被移除，如下：

 。

我们说Surround With是Surround with Live Template的超集，意思是Surround With包含了Surround with Live Template的功能，如，我们有一个names为变量名的集合，输入这个变量名，然后按下Ctrl + Alt + T，上面是代码块，下面则是 Live Template模板，如下：



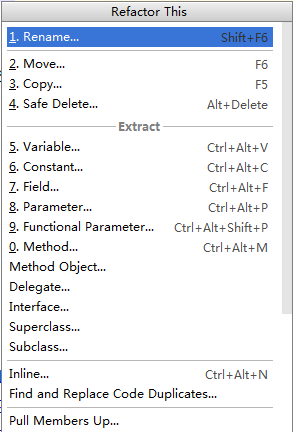
# 第四章 代码重构

重构代码的最大风险是可能会引入意外的错误，AndroidStudio通过分析某些具有危险性的重构操作来降低这些风险，并直接激活Find工具窗口，其中标识出了所有的错误和冲突，使你可以在提交之前预览所做的修改。尽量不要使用蛮力来重构，如全局查找替换，这种情况下AndroidStudio无法保证不会引入错误。如果使用AndroidStudio来重构，则AndroidStudio会帮你分析有哪些危险并处理掉，如在Project工具窗口中将某个Java源文件从一个包拖到另一个包中会触发一次Refactor🡪Move操作，它会分析移动操作所产生的影响，让你预览修改，并接着优雅地将整个项目中所有针对该类的import语句修改为新包名的完整路径。

大多数的重构操作都局限于一个方法或类中，因此不太可能向项目中引入错误。有风险的重构操作是指那些涉及两个或更多个资源的操作。如果重构操作引入了编译错误，Inspections Manager将会在Editor中使用红色标签标识出受影响的资源。在这种情况下，可以尝试修改它们，或者通过按Ctrl + Z来简单地撤消整个重构操作。如果重构操作成功且没有编译错误，但却涉及大量资源，那么仍然需要进行测试，以验证没有引入任何运行时错误。第11章涵盖了有关测试的内容。

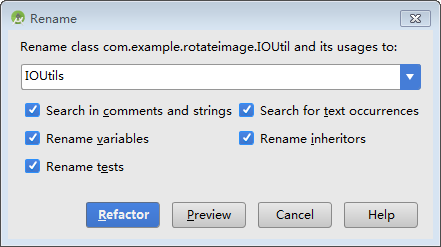
提示： 应该将所有重要的重构修改单独做一次Git提交，以便之后可以轻松地回退。第7章涵盖了有关git的内容。

在开始探讨个别重构操作之前，我们需要指出AndroidStudio有一项极其便利的重构操作，称为Refactor->Refactor This，此操作会显示一个上下文菜单，其中囊括了最有用的重构操作，它对应的快捷键为：Ctrl + Alt + Shift+ T，重构的命令很多，如果记不住的话记这一个就够了，界面如下：



## 4.1 重命名

在Project工具窗口中选中某个.java文件，按下Shift + F6，出现的对话框允许重命名类，以及该名称在注释、测试用例和继承类中所有出现的地方



在代码中的类上定位一下，然后按Shift + F6，这样的方式也可以重构类名，但是不会出现上面的对话框。

Search in comments and strings：在注释中和代码的String字符串中，如果出现与类名一样的字符，则也可以重命名

Rename variables：重命名量名

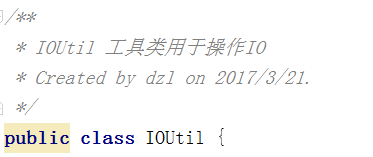
Rename tests：重命名测试用例

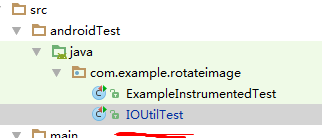
Search for text occurrences：暂不知道干嘛用的

Rename inheritors：重命名继承者的类名

演示如下：

在项目中有IOUtil.java文件，如下：



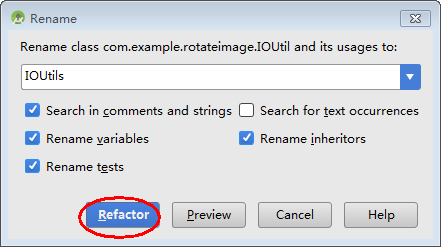
有该类的测试用例类，如下：  


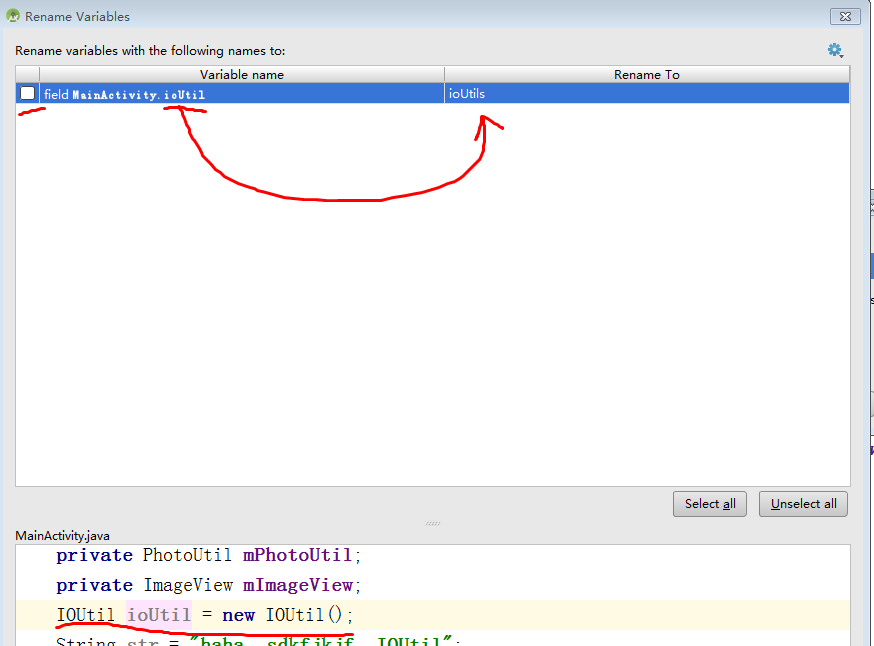
有该类的继承类：  


在MainActivity中有使用到MyUtil，并且在String中有IOUtil的字符：

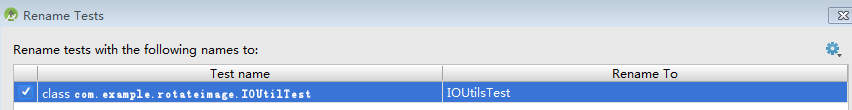


在Project工具窗口中选中某个IOUtil.java文件，按下Shift + F6，勾选如下：

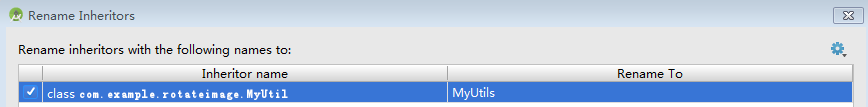


接下来将会出现一系列的对话框来显示可重构的地方，如下：  


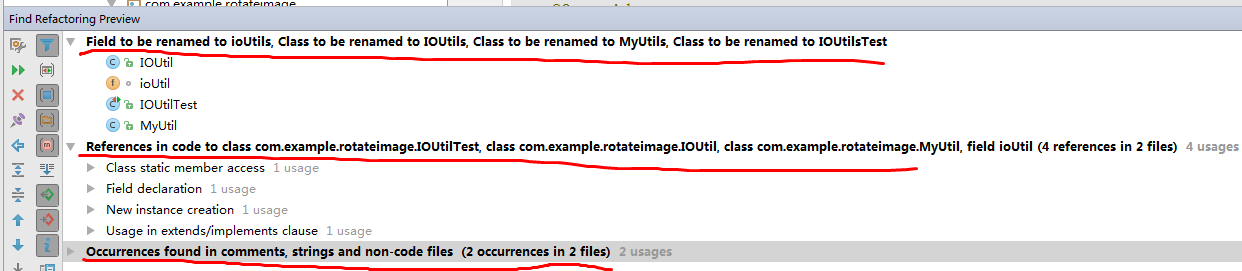
标题告诉我们这个操作是要Rename Variables，下面有一个复选框，打上勾代码要重命名，不打勾则不会，上图中显示字段MainActivity.ioUtil会被重命名为ioUtils，下方窗口有显示IOUtil 变量的源代码。点击OK，将出现下一个对话框，如下：



这是重命名测试用例的类名，把IOUtilTest重命名为IOUtilsTest，点击OK，出现下一个对话框，如下：



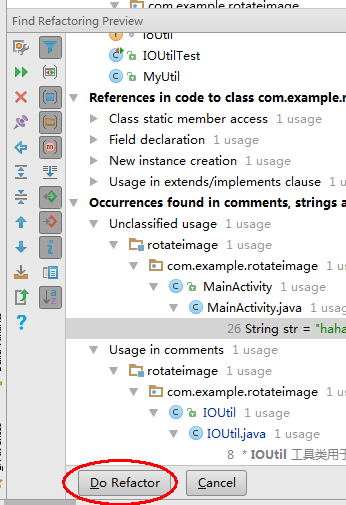
这是重命名继承类的类名，把MyUtil重命名为MyUtils，点击OK，这时不会再出现对话框了，出现了一个重构预览界面，如下：



上图中画了3条红色，中间红色的内容不需要看，它是把引用到IOUtilTest、IOUtil、MyUtil、变量ioUtil这4个我们需要重构的东西所在的地方给我们列出来，这个看不看无所谓。主要看第一行红色，它告诉我们字段会被重命名为ioUtils，类会被重命名为IOUtils，类会被重命名为MyUtils，类会被重命名为IOUtilsTest，其实就是告诉我们哪些东西会被重构。 第3天红色线地方也是我们需要看的，这个地方是告诉我们在注释和代码的字符串中找到了IOUtil单词，这些单词也会被重命名，可展开查看具体内容，如下：

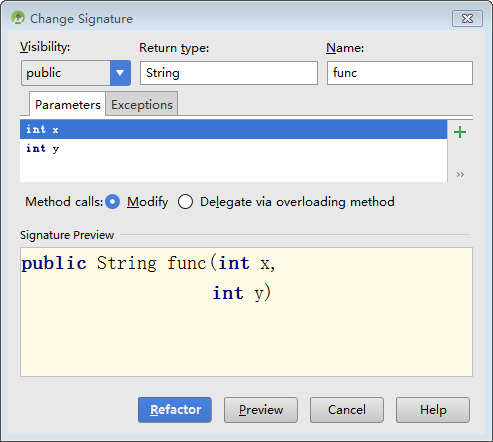


这告诉我们这两个地方的字符串也会被修改，双击红色行可跳转到这些字符串所在源文件位置。最后点击“Do Refactor“按钮执行重构，如下：



## 4.2 修改签名

快捷键：Ctrl + F6 ，可以修改方法的可见性、名称、返回类型、参数和抛出异常。把光标定位在方法名上，然后按下Ctrl + F6(Refactor🡪Change Signature)，出现如下对话框：

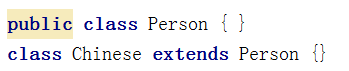


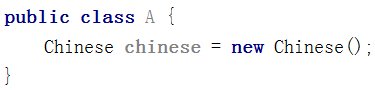
在Method calls选项中，如果选择“Delegate via overloading method”，它是给你生成一个新方法，对原来的方法没有任何影响。

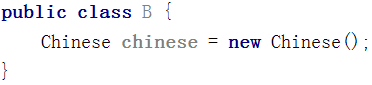
## 4.3 类型迁移

Refactor🡪Type Migration允许把一种Java类型改为另一种。假设创建了Chinese类，如果想要将Chinese的所有实例修改为Person，那么Type Migaration很容易实现这一点。

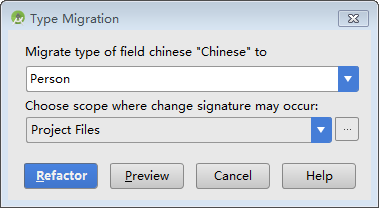
有如下代码：







现要将A和B中的Chinese实例换成Person，在A类中的Chinese上定位光标，然后按下Ctrl + Shift + F6，出现如下对话框：



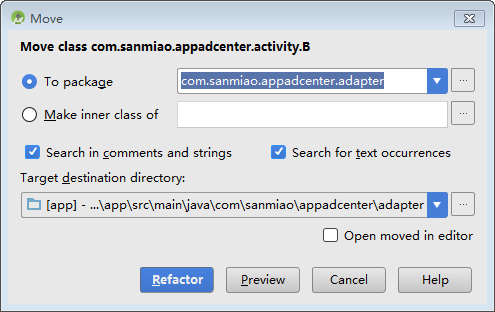
目前发现这个功能并不管用，只能把当前的一个Chinese变成Person。

## 4.4 移动

有三种方法可以移动源文件：

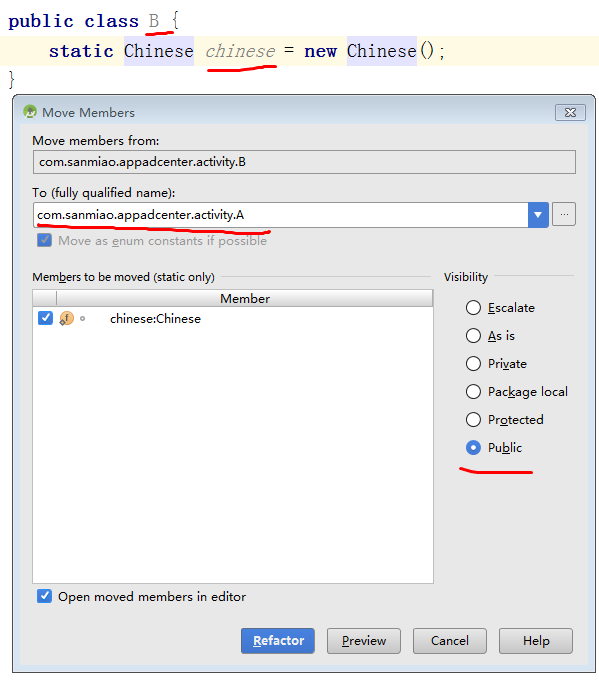
1. 在Project窗口中把源文件从一个包拖曳到另一个包下
2. 选择该源文件并从主菜单中选择Refactor🡪Move
3. 在Project窗口中选择文件，并按F6键

这些操作都会弹出如下窗口：



如果选择Make inner class of，则可以把选择的类变成另一个类的内部类

除了移动类，还可以移动静态的成员变量，把一个类的静态成员变量变成另一个类的静态成员变量，在静态成员变量的变量名上定位，然后按F6，如下：



如上图，我们把B类中的静态成员变量chinese移到了A类中，并且把可见性设置为Public，点击Refactor之后则B类中的chinese成员变量则没有了，变到A类中去。 可见性中的“Escalate”和“As is”暂时不知道起什么效果。

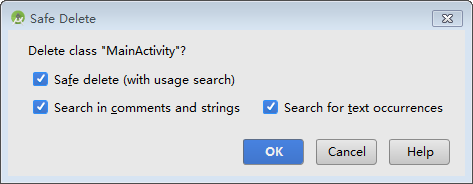
## 4.5 复制

有3种方式

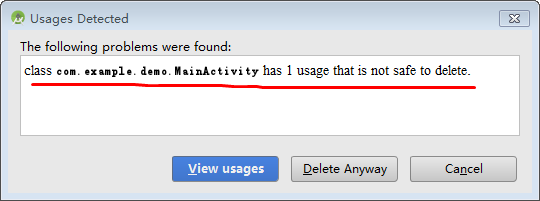
1. Ctrl + C与 Ctrl + V
2. 选择文件🡪Refactor🡪Copy
3. 选择文件🡪F5

## 4.6 安全删除

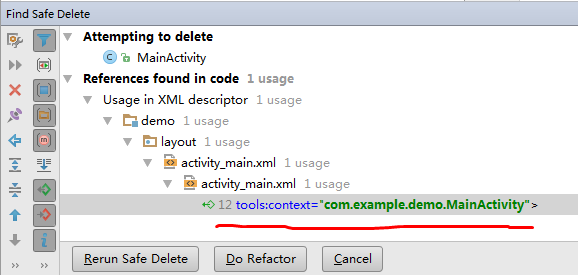
在Project窗口选择MainActivity，按下Delete键，弹出如下对话框：



点击OK后出现如下对话框：



上图红线意思是MainActivity有一个使用不能安全删除，点击Delete Anyway表示不管如何都要删除，点击ViewUsages可查看有什么使用，如下：

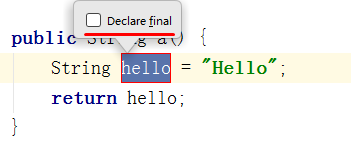


这说明在activity\_main.xml中有对MainActivity的依赖，点击Do Refactor执行删除。

## 4.7 抽取

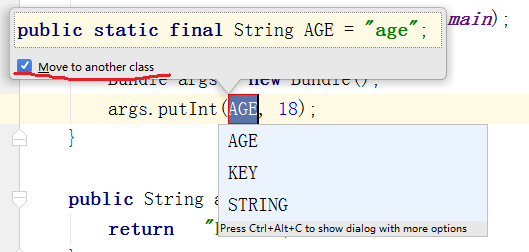
### 4.7.1 抽取局部变量

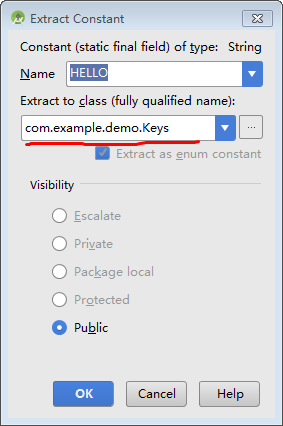
Ctrl + Alt + V (Variable)，在出现的对话框中可以选择是否要抽取成final类型的局部变量，如下：



### 4.7.2 抽取常量

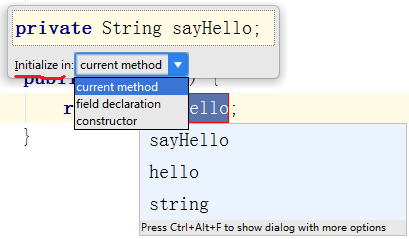
Ctrl + Alt + C (Constant)，在出现的对话框中可以选择这个常量是否要声明到其它类中去，如下：





### 4.7.3 抽取字段

Ctrl + Alt + F (Field) ，在出现的对话框中可以选择初始化字段的位置，如下：



Current method：在当前方法中进行初始化

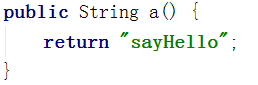
Field Declaration：在声明字段的地方进行初始化

Constructor：在构造方法中进行初始化

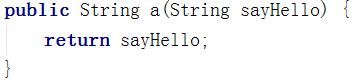
### 4.7.4 抽取参数

Ctrl + Alt + P (Params)，把方法中的局部变量或字段抽取成参数。

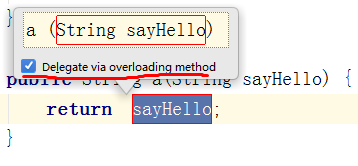
有如下方法：

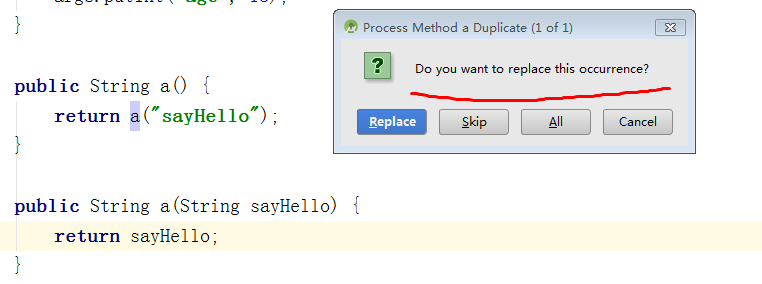


对“sayHello”进行参数抽取，效果如下：



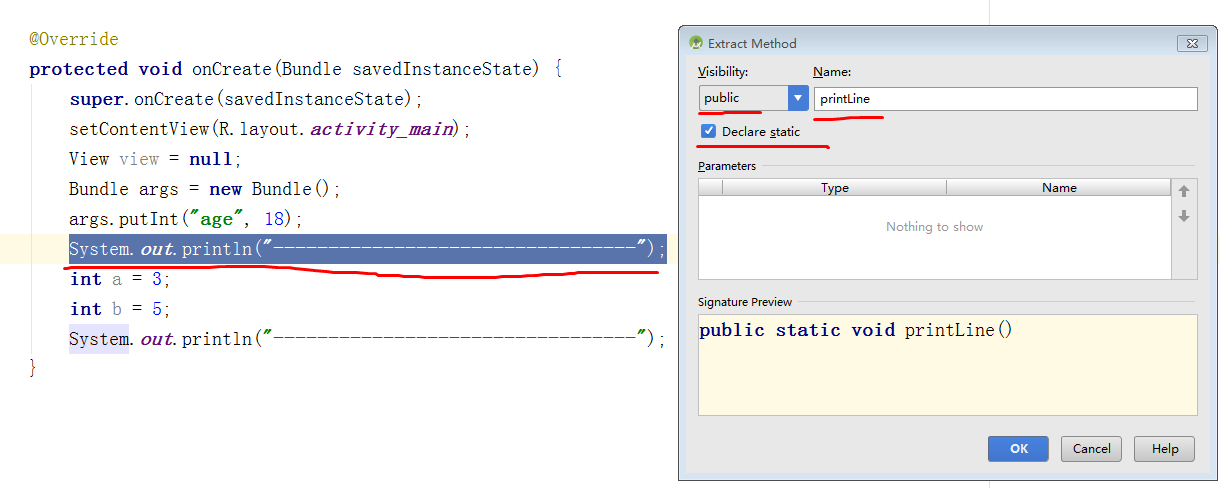
在抽取的时候，如果勾选“Delegate via overloading method”，则生成一个重载方法，原方法不变，如下：



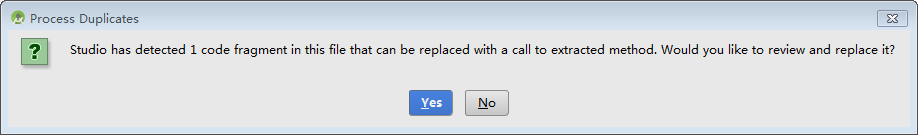


上述对话框中，点击“Replace”、”Skip”、”All”发现效果一样，不知道有什么区别。

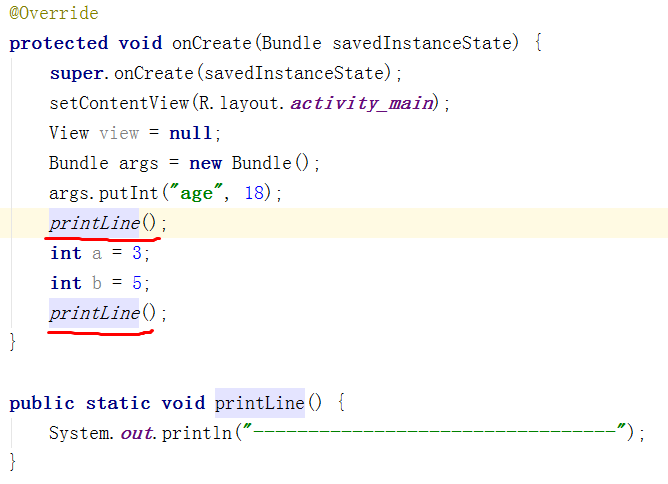
### 4.7.5 抽取方法



“Visibility”选择方法的可见性，”Declare static”把方法设置成静态，点击上图的OK，出现如下提示：



大概意思为Studio中有1个代码片断和我们现在抽取的代码片断是一样的，可以使用我们抽取的方法替换这个代码片断，你想要检验并且替换它吗？我们点击Yes，效果如下：

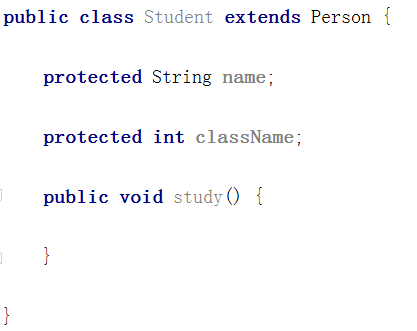


## 4.8 高级重构

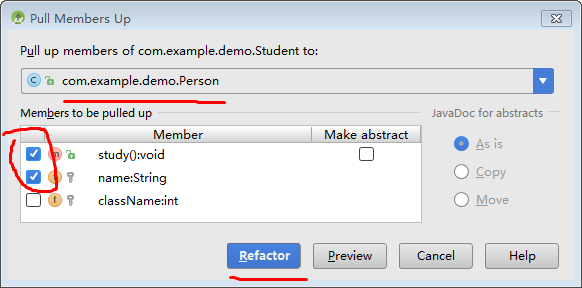
### 4.8.1 下推成员/上拉成员，成员包括字段和方法

有如下两个类，Student继承了Person：

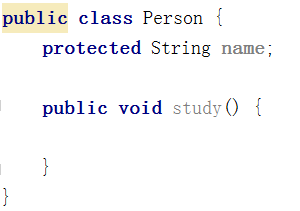


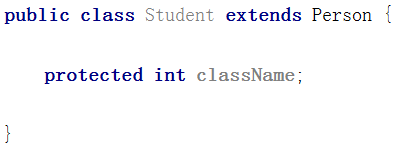


我们发现，人都有名字，人都会学习，所以Student类中的name成员变量和study 成员方法应该属性Person类，把光标定位在Student类的任意位置，执行：Refactor🡪Pull Members Up，出现如下对话框：



选择要提升到的父类（Person），并选勾先需要提升的成员（studey与name），然后点击Refactor，效果如下：

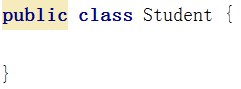




如果觉得父类中的某些成员只属于某个具体的子类，则执行：Refactor🡪Push Members Down

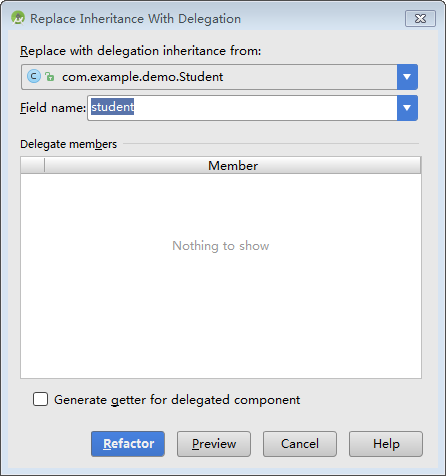
### 4.8.2 使用代理代替继承

有如下两个类：

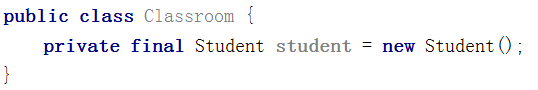




我们发现Classroom不是Student，而是包含Student，要修改这个关系，先把光标定位在Classroom类的任何位置，然后执行：Refactor🡪Replace Inheritance with Delegation，效果如下：

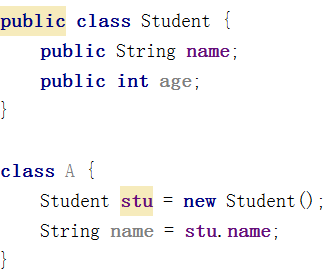


这里可以给Student字段起一个名字，最终效果如下：

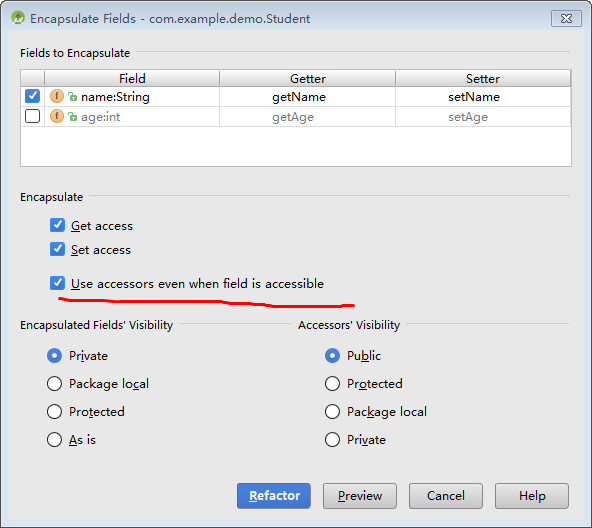


### 4.8.3 封装字段

封装是一种面向对象策略，它通过将字段的访问级别设为私有来隐藏它们，接着通过公共可性的getter/setter方法为这些字段提供公开接口。Refactor🡪Encapsulate Fields类似于Code🡪Generate🡪Getter and Setter，只不过是Encapsulate Fields提供了更多的选项，把光标定位在类的任意位置，然后执行：Refactor🡪Encapsulate Fields，Encapsulate单词意思为封装。效果如下：



点击Refactor按钮后，效果如下：

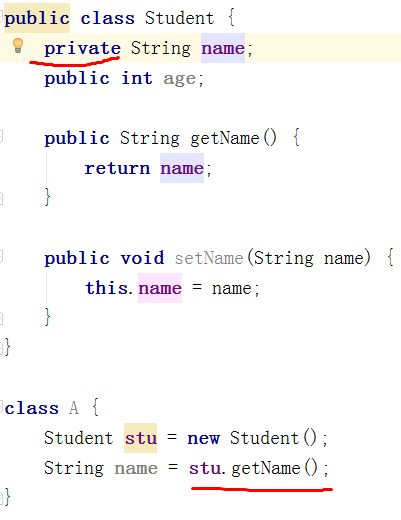


选择需要封装的字段， Encapsulate有3个选项：

* Get access 产生getter方法
* Set access 产生setter方法
* Use accessors even when field is accessible 不知道干嘛用的

Encapsulate Fields’ Visbility 可以修改字段的可见性，As is的意思为照现在的样式，即当前字段的可见性保持不变

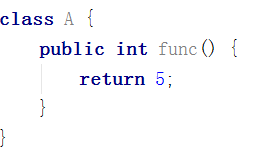
Accessors’ Visibility 可以修改setter/getter方法的可见性



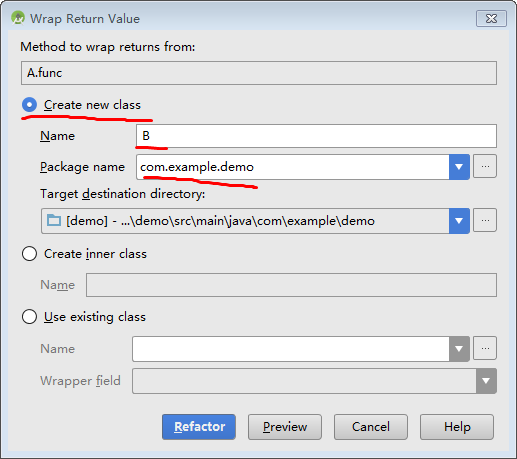
从上面结果中红色画张中可看出，Encapsulate Fields功能可以修改原字段的可见性，并且会把调用了name属性的地方改为调用对应的getter方法，而Code🡪Generate🡪Getter and Setter是不能修改字段可见性的，也不会把调用了name属性的地方改变getter方法调用。

### 4.8.4 封装方法返回值

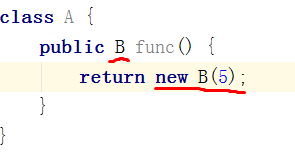
Refactor🡪Wrap Method Return Value

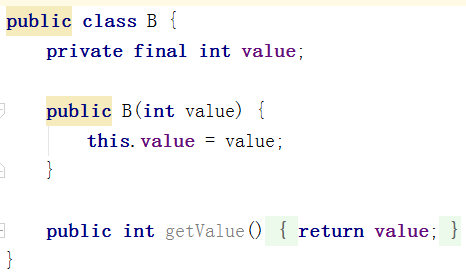


在func方法中定位一下，然后执行重构命名，出现如下对话框：



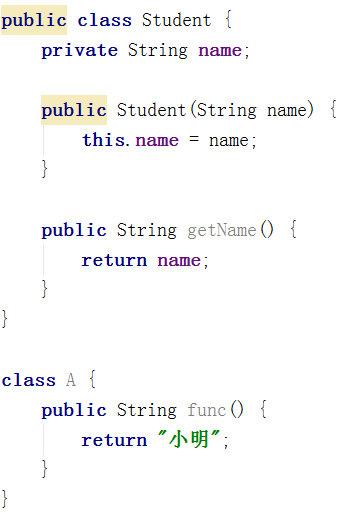
包装方法返回值有3种方式，Create new class方式为创建一个新类来包装返回值，点击“Refactor”按钮，效果如下：



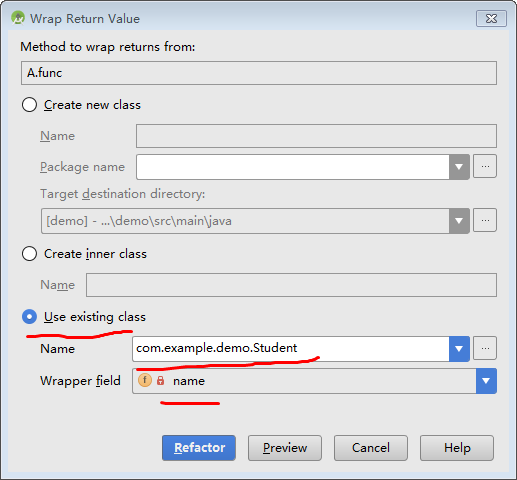


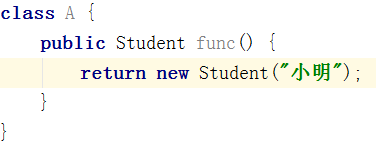
Create inner class为创建内部类来封装方法返回值

Use existing class 为使用已经存在的类。 从生成的自动生成的B类的截图来看，如果要使用已经存在的类，则结果要与上面的B类相似，即要有对应类型的字段，且要有接收这个字段的构造方法，并且要有获取这个字段的getter方法，如下：



在func中的返回值可用已经存在的Student类进行包装，效果如下：

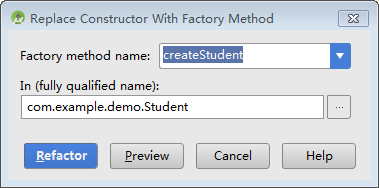


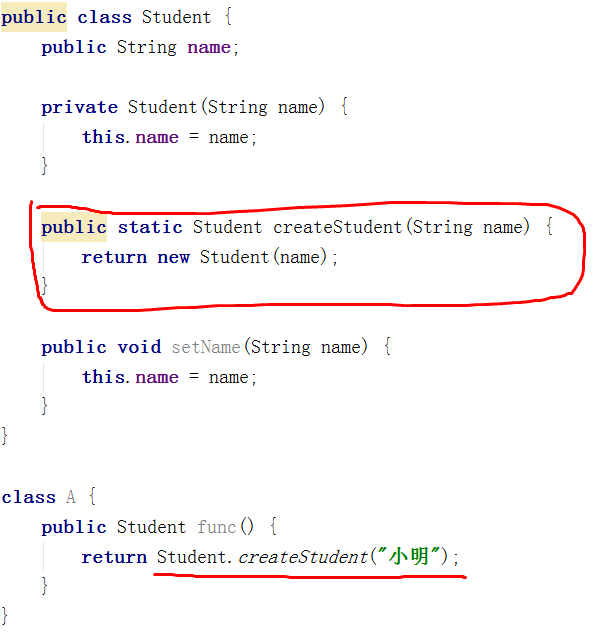


### 4.8.5 使用工厂方法代替构造函数

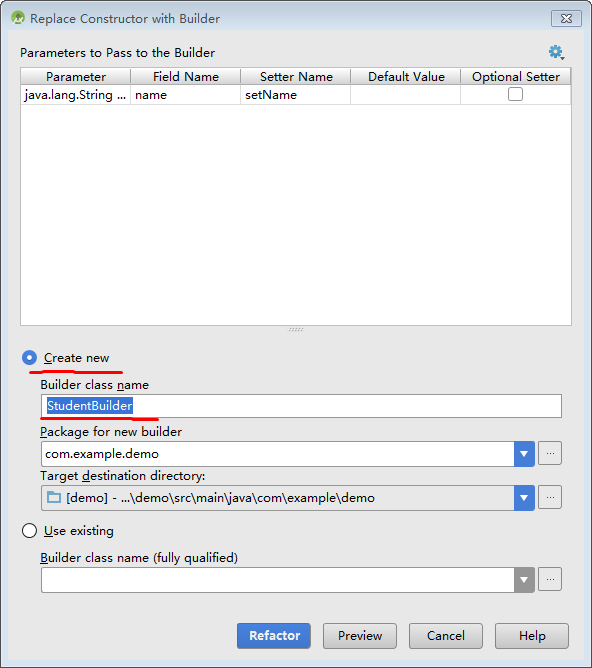
有如下类：

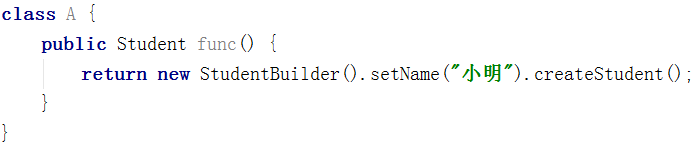
把光标定位在构造方法名上，然后执行Refactor🡪Replace Constructor with Factory Method，如下：

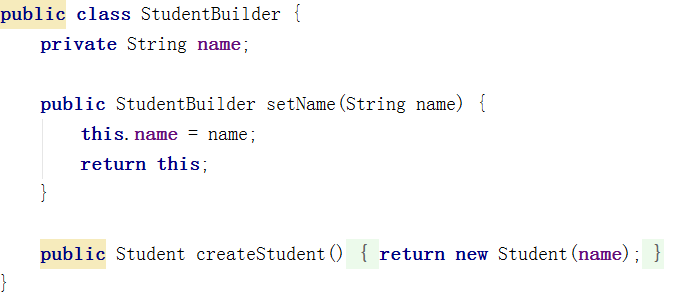




有一个类似的命令：Refactor🡪Replace Constructor with Builder，效果如下：



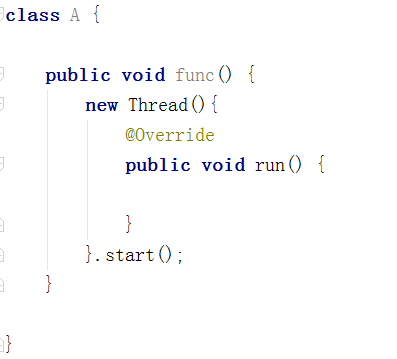




上面对话框还有一个“Use existing”的选项，可以选择使用已经存在的Builder类来替换，类的内部结果要与上面的StudentBuilder类的内部类似如Stuent的构造方法接收的参数名是name，则要求你的Builder类中有name属性与setter方法，和createStudent方法，如果没有，则AndroidStudio会自动创建类似的结构。

### 4.8.6 将匿名类转换为内部类

有如下代码：



把光标定位在匿名类的任意位置，然后执行：Refactor🡪Convert Anonymous to Inner，效果如下：

