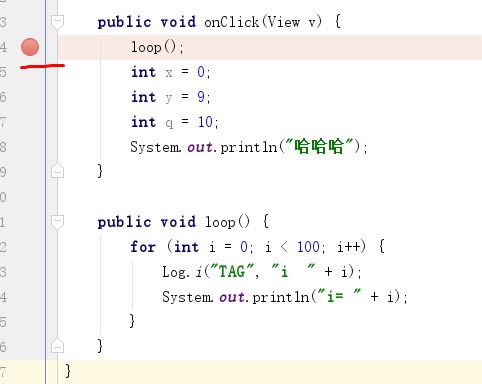
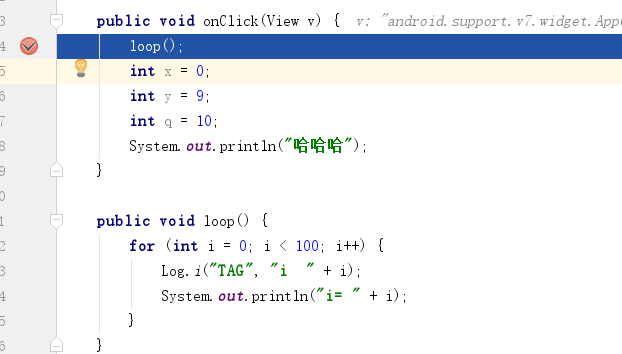
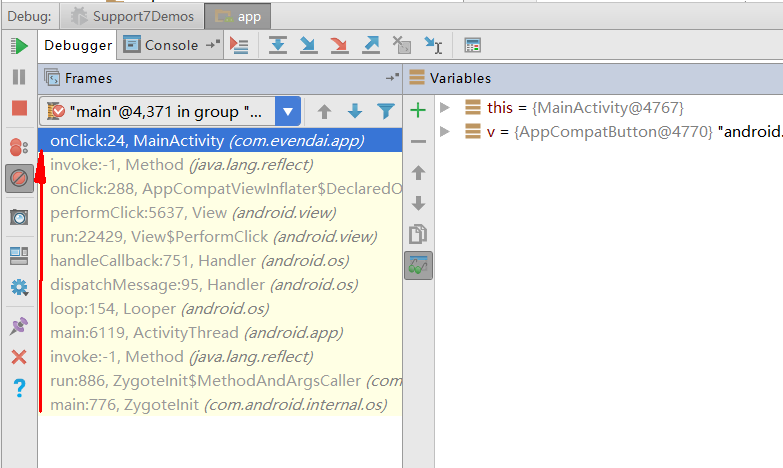
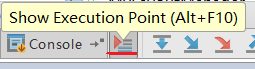
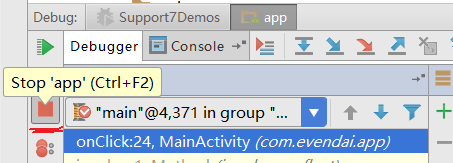
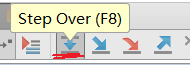
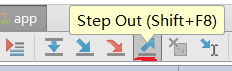
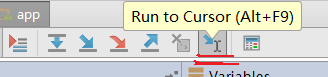
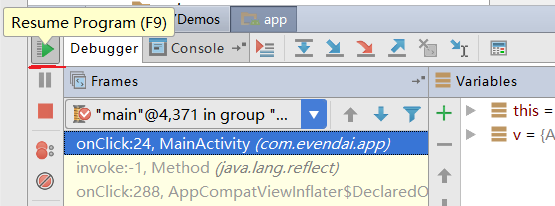
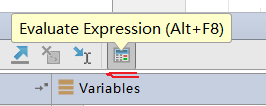
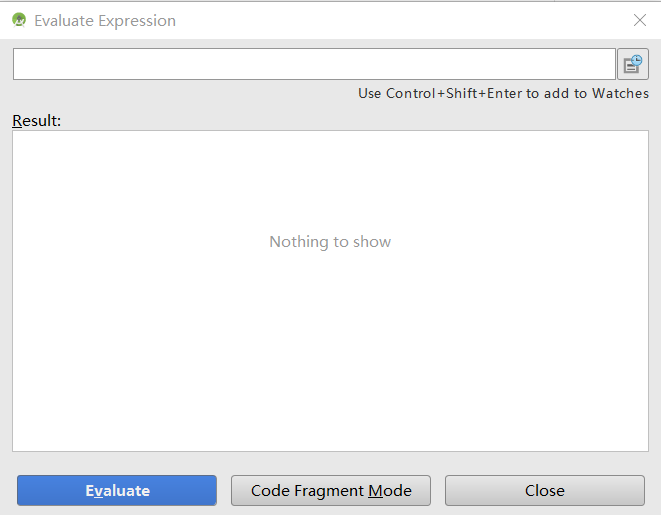
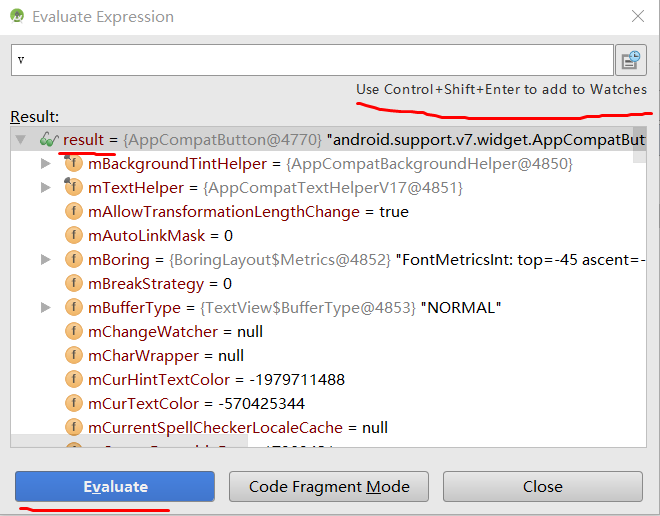
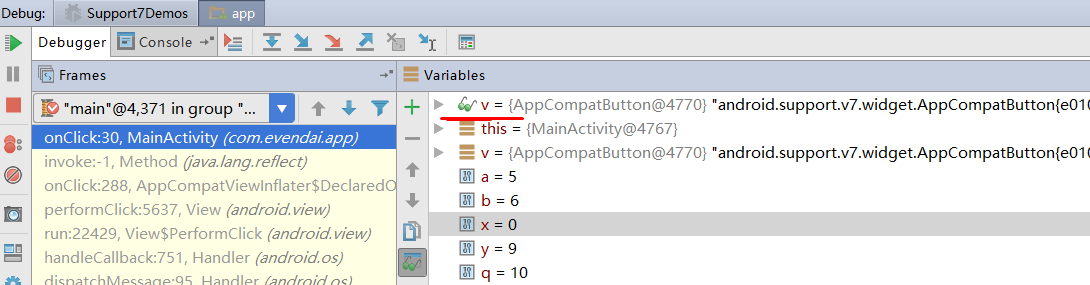
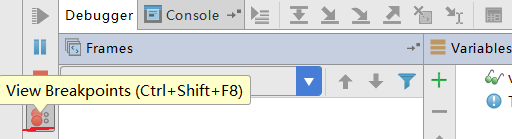
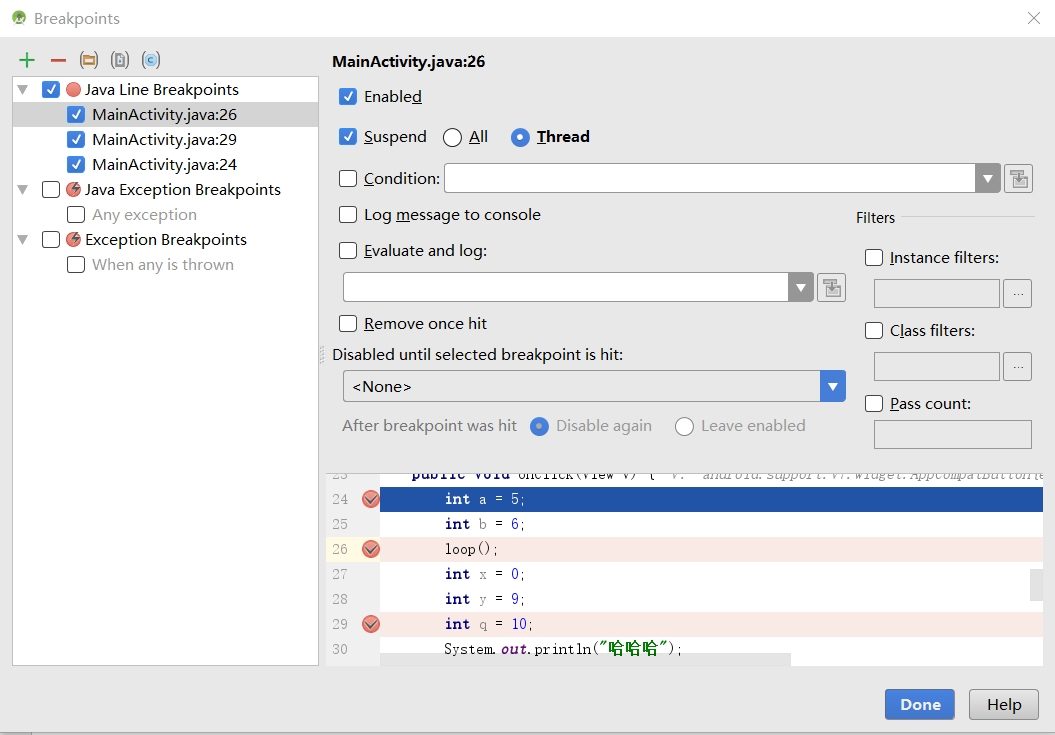
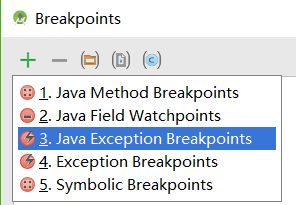
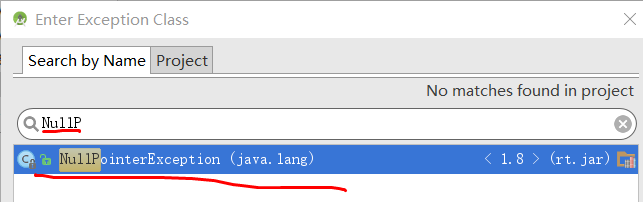
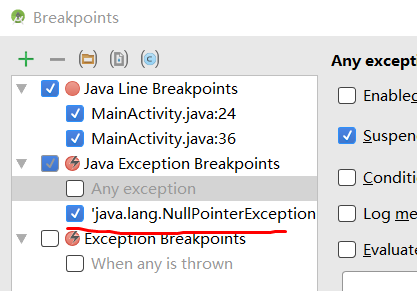
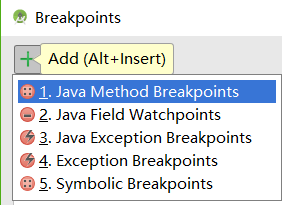
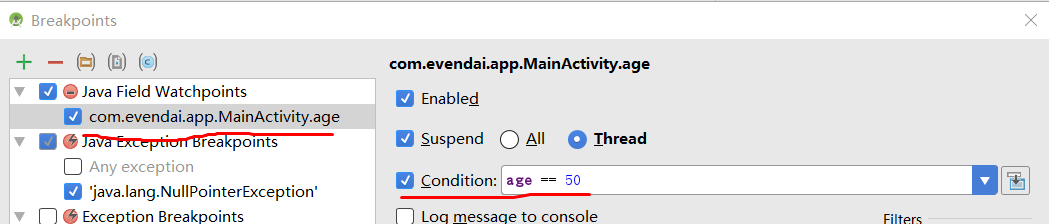
1. 代码中打上断点，如下：  
    
2. 以Debug方式运行：  
    、
3. 执行到断点代码的时候会出现深蓝色条，表示断点执行到的代码，如下：  
      
    这个时候屏幕的底部会出现Debug调试窗口，如下：  
      
    这里左边窗口显示了onClick的调用流程（因为断点打在了onClick方法里面），右边显示方法中变量的值（包括方法参数）  
     
      
    Show Execution Point（显示执行点），其实就是给你定位到当前执行到的那一行代码处。比如你滚动了屏幕，想快速回到现在代码执行到的位置则可以点击这个按钮。  
     
      
    Stop ‘app’（停止app），其实就是把app进程杀了，调试也会跟着终止  
      
      
    一条一条的执行代码，Over有“越过”之意，在执行到方法调用时不进入方法体，这就是所谓的“越过”，把方法的调用当成是一条代码。  
     
      
    一条一条的执行代码，Into有“进入”之意，即执行到方法调用时会进入方法体，这就是F7与F8的本质区别  
     
      
    Out有出去之意，即把整个方法里的代码执行完，然后就出去到外面了。  
      
      
    Run to Cursor（运行到光标），这个比较容易理解，在某一行代码上点击一下，即定位光标，这时点击这个按钮的话即直接运行到光标所在代码，但是需要注意：如果光标之前的代码有断点，则会跳到之前的断点，所以，这个按钮也可用作跳到下一个断点的功能。  
     
      
    Resume Program (F9) ，恢复程序，执行代码到下一个断点，如果没有断点了则执行完所有代码。  
     
      
    Evaluate Expression（评估表达式，一个变量也是表达式），点击这个按钮后出现如下对话框：  
      
    在这里，我们可以查看所有已经运行过的代码中的变量，比如有变量v，则输入v，然后点击“Evaluate”按钮，效果如下：  
      
    result即我们变量v的值，我们发现这个变量v是一个AppCompatButton的对象。上面有一句提示按Ctrl + Shift + Enter可以把当前变量添加到Watches，如下：  
      
    添加Watches也可以在代码中选中表达式，然后右击选择“Add to Watches”  
     
      
    View Breakpoints（查看断点），点击后可看到所有的断点，如下：  
      
    左边显示的是断点列表，有3个断点分类：Java Line Breakpints（Java行断点）、Java Exception Breakpoints（Java异常断点）、Exception Breakpoints（异常断点）。右下角显示了断点所在的代码。  
    异常断点如何添加，左上角的“加号”，选择如下：  
      
    然后在出现的对话框中输入空指针异常，如下：  
      
    点击OK后结果如下：  
      
    当程序发生空指针异常时，就会进入Debug模式。如果希望任意异常都进入Debug，则可以在“Any exception”中打勾即可。  
     
      
    从这里我们可以看到，其实断点的类型有这么多，比如添加 一个Field，即一个成员变量，当这个成员变量的值发生变化时则进入Debug模式。也可以设置条件，如下：  
      
    当MainActivity中的age等于50时进入debug模式，否则不进入。