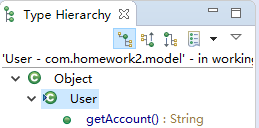
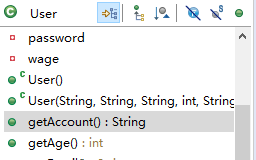
**Eclipsed的Debug调试**

**一、Type Hierarchy窗口：**

1.在Eclipse中按Fn+F4可以快速打开类型层级Type hierarchy窗口。这个Type Hierarchy窗口可以显示当前类的继承关系以及显示类的信息，如属性、方法、构造器。

①Type Hierarchy窗口





红色方框中各表示的含义：

第一个是Lock View and Show Members in Hierarchy：锁定当前窗口中的类，并展示当前类的所有成员(成员属性、成员方法、构造器)，不包括继承的类中的成员。

第二个：Show All Inherited Members：展示当前类的所有成员，包括继承父类的成员。

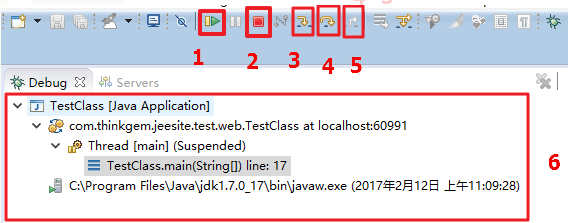
第三个：Sort Methods By Defining Type：对当前类中的方法进行定义类型排序显示。

第四个：Hide Fields：隐藏成员属性，把构造器和成员方法显示出来。

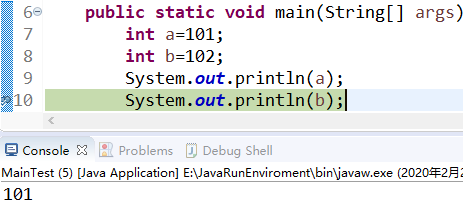
第五个：Hide Static Fields and Methods：将静态成员属性和静态方法隐藏，将其它显示出来。

第六个：Hide Non-Public Members：将不是public 修饰的成员隐藏，将其它显示出来。

**二、DebugView(线程堆栈视图)窗口：**



当使用debug模式运行程序时，**程序会运行到断点处然后暂停(断点处的代码不会执行)**，然后等待我们的下一步运行的指令，如图：直接Debug运行代码，控制台上只输出了101，而102没有输出。



**①：Resume(F8)：**表**从暂停处继续执行代码，直到下一个断点处暂停**。运行到当前方法的下一个断点；若当前方法调用了其它方法，则运行到调用方法的断点处暂停。若调用的其它方法没有断点则会直接将调用方法运行完，直到遇到下一个断点暂停。Resume指令都是直接将代码从一个断点处运行到下一个断点处，不会一步一步的显示代码的执行过程。

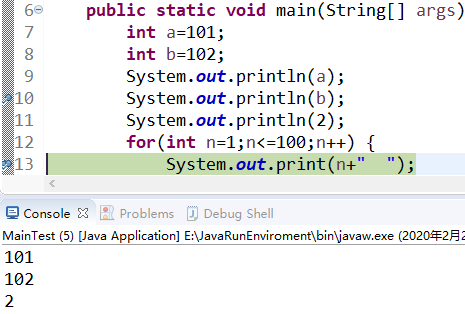
**②：Terminate：**结束当前程序的运行和调试。

**③：Step Into(F5)：单步执行，点击一下，就执行一次，遇到方法中调用其它方法时，也会跳到调用方法中进行单步执行**。

**④：Step Over(F6)**：**在主方法中单步执行，若主方法调用了其它方法，则不会进入调用的其它方法中单步执行，而是直接把调用的其它方法当成一步来执行**。在调用其它方法的执行过程中这是和Step Into的执行的区别。**这个常用**。

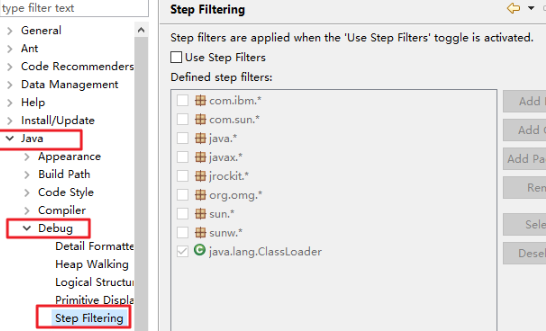
**⑤：Step Return(F7)**：**只有进入了被调用方法的执行中，该按钮才有效**。即只有使用了Step Into按钮然后进入了被调用方法内Step Return方法才有效，若没进入则是灰色的不可用。**使用该Step Return可以直接将被调用方法的代码当作一步执行，执行完之后跳出该方法然后返回到主方法(调用该方法的方法中)中**。

将下面代码Debug运行，先直接在控制台上输出101，然后暂停了。当点击Resume按钮后，运行过程直接从第10行代码跳到第13行代码，然后直接在控制台上输出结果为102和2。执行了for循环n=1，但是没有执行输出n==1。

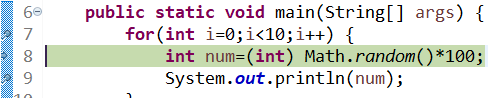


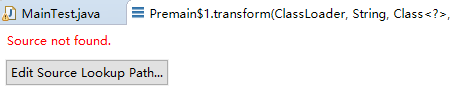
**⑥Step Filters：**在Debug的时候，当想要忽略一些我们不关注的类(如Debug的时候有时候会进入源码中进行单步执行，我们想忽略对源码的操作时)，这时我们可以可以开启Step Filters进行过滤。

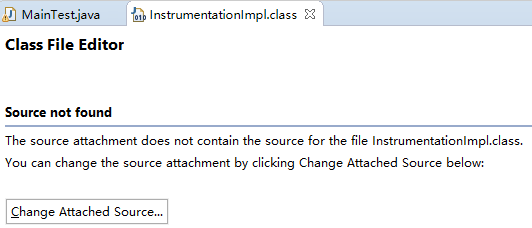
设置Step Filters：Eclipse的Windows-->Preference-->Java--> Debug-->Step Filter。然后勾选Use Step Filters，然后勾选自己需要过滤的包或者自己添加包，然后点击Apply即可。



例子：(没有点Step Filters)当程序运行到这的时候，点击Step Into进入下一步，则进入到Math.random()方法内容进行单步执行 (若点Step Over则不会进入，因为将Math.random()方法当作一步执行了)，出现下面这个页面。 若设置了Step Filters，即使是点击Step Into也不会进入源码页面对源码进行单步执行。

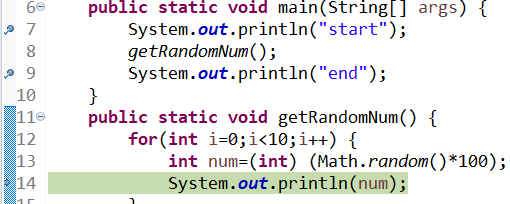






**⑦Drop to Frame**：**当进入某个被调用的方法内时，点击Drop to Frame可以让程序回到该方法的开头第一行重新开始执行。**若没有进入被调用的方法内，该按钮是灰色的，无效。

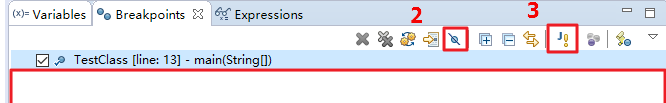
例子：当单步执行进入了getRandomNum()方法内部操作时，如操作到当i==5时，点击Drop to Frame按钮，则重新回到该方法的第一行且i==0。



**三、Variables View(变量视图)**

在变量窗口，选择窗口中的变量，右键点击Change Value 可以改变变量的值。如for循环中可以将i改为想要的值如i=50。Copy Variables可以可以复制变量的值，尤其是当变量的值很长时，如json格式的字符串。Find可以查找变量。

**四、Breakpoints(断点视图)**



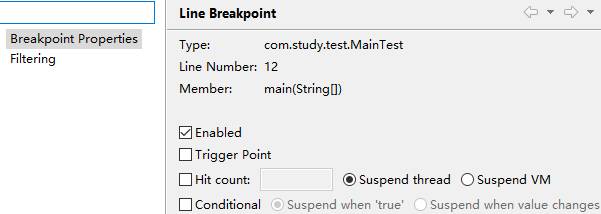
按钮2可以将窗口中的断点失效。

按钮3异常断点，当出现异常时可以添加指定类型异常的断点。

**三、设置断点**：

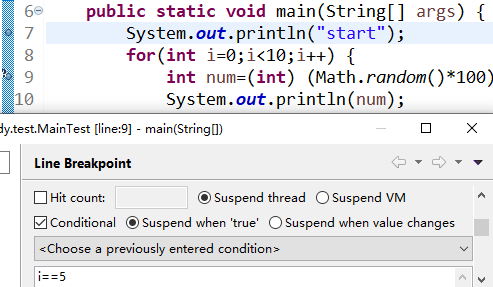
**1.行断点(Line Breakpoints)**：条件断点。**在各断点设置条件，只有满足了用户设置的条件，代码才会在运行到断点处时停止**。

将鼠标指针指向代码中左侧的断点，右键然后点击Breakpoint Properties，进入断点的设置。如图：



只要代码打了断点默认都是Enabled，即表该断点有效，启用了该断点。Hit count表执行多少次之后在该断点处暂停，一般用于循环语句中，如指定Hit Count值为10，那么当前的循环执行完第9次的时候在该断点处暂停。Conditional即条件判断，若需要循环变量i==5时，线程挂起，则条件设定为i==5，选择Suspend when true。

例子：下面程序启动之后，点击Resume，则程序会运行到i==5之后暂停。注意下面条件判定的断点是打在int num这一行，而不是打在for()循环这一行。



若上面的Hit Count和Conditional都选择的话，如果表达式和值设置不合理则会失效。如果选择Suspend when value changes，那么可能当Conditional在变量值发生改变的时候就挂起。

**2.观察断点(watch Breakpoints)**：变量断点。**将断点打在成员变量处**。**只有对象的成员变量有效果，静态成员变量不起作用。**

**3.方法断点(method Breakpoints)**：将断点打在方法上。**方法断点的特别之处在于它可以打在 JDK的源码里**。

**4.异常断点(exception Breakpoints)**：在异常处添加指定异常类型的断点。异常断点可以点击Breakpoint View窗口中的按钮进行添加。

**5.类加载断点(class load Breakpoints)**：在类名上打的断点。接口上是打不了类加载断点的，但是抽象类是可以的。类加载断点不管是打在抽象或者非抽象类上，都会在类第一次加载或者第一个子类第一次被加载时，挂起线程/VM(没理解)。