实验 2：软件静态测试实战

董瑞志
常熟理工学院

实验目标

了解代码坏味道的定义、类型；
了解编码风格的重要性，掌握提高代码可读性的常用策略。
掌握常用的程序度量指标；
熟练使用 SourceMonitor 进行代码度量；
运用编码规范、代码检查表、程序静态分析工具进行程序静态测试。

课时安排

4 课时

相关知识

1 软件静态测试。
2 常用的软件静态测试技术包括：
1） 针对程序制品的桌面检查、代码走查、代码评审、程序静态分析；
2） 针对包括程序制品在内的各种软件制品开展的正式设计评审。
3 程序静态分析，包括词法分析、语法分析、抽象语法树分析、控制流分析、数据流分析、污点分析，等等。

内容安排

第 1 节 代码坏味道

代码坏味道的提出是为了更加直观地表示出代码中潜在问题，期望通过软件重构予以改善。代码坏味道的类型包括应用级坏味道、类级坏味道、方法级坏味道。

1.1 应用级坏味道

1） 重复代码（ duplicated code）。

重复代码是指在一个以上的地方出现了重复的代码片段。重复的代码结构使程序变得冗

长，需要重构。例如:

①同一个类中两个函数使用了相同的表达式。解决方案:使用提炼方法（Extract Method）提炼出重复的代码，让两个函数同时调用这个提炼方法，而不是重复使用相同的表达式。

②一个父类有两个子类，这两个子类中存在相同的表达式。解决方案:对两个子类使用提炼方法，然后使用上移方法（Pull Up Method）将提炼出来的代码定义到父类中。

2） 人为复杂性（ contrived complexity）。

在进行简单设计就能解决问题的情况下，强制用了过于复杂的设计模式。

3）散弹式修改（ shotgun surgery）

散弹式修改则是指一种变化引发多个类发生相应修改。

如果需要修改的代码散布四处，你不但很难找到它们，也很容易忘某处重要的修改。这种情况应该使用移动方法（ Move Method）和移动域（Move Field）方式把所需要修改的代码放进同一个类中。如果当前没有合适的类可以安置这些代码，创造一个。也可以运用内联类（ Inline Class）把一系列相关行为放进同一个类。这可造成少量分散变更，但你可以轻易处理它。

1.2 类级坏味道

1） 过大的类。

如果单个类做太多事，类中就会出现太多实例变量、拥有大量的方法，造成过大的类。

可以使用 Extract Class 将几个变量一起提炼至新类内提炼时应该选择类内彼此相关的变量，将它们放在一起。有时候，类并非在所有时刻都使用所有实例变量。或许可以多次使用 Extract class 或 Extract Subclass

如果过大类是一个 GUI 类，可能需要把数据和行为移到一个独立的领域对象中。也可能需要两边各保留一些重复数据，并保持两边同步。

2） 依恋情结的类。

即一个类的方法过多地使用了另一个类的方法。也可以说，某个类的函数对另一个类的兴趣高过对自己所处的类。

通常的焦点就是数据，如某个函数为了计算某个值，从另一个对象中调用几乎半打的取值函数。这时应该运用 Move Method 把它移到适当的地方。有时候函数中只有一部分受这种“依恋之苦”，这时候使用 Extract Method 把这部分提炼到独立函数中，再使用 Move Method 带它去它的“梦中家园”。

3） 过度亲密类。

一个类存在与另一个类实现细节的依赖关系。也就是说，两个类过于亲密，花费太多时间去探究彼此的 private 成分。

可以采用 Move Method 和 Move Field 划清界限，也可以通过把双向关联变成单向关联的方法让其中一个类对另一个类“断情丝”。如果“情投意合”，可以运用 Extract Clas 提炼到一个安全地点。也可以使用隐藏的委托人（hide delegate）传递“相思情”。如果让一个子类独立生活，请使用委托人机制来代替继承机制。

4） 拒绝的馈赠。

子类仅仅使用父类中的部分方法和属性，其他来自父类的“馈赠”则成为了累赘。

问题的原因是有些人仅仅是想重用父类中的部分代码而创建了子类，但实际上父类和子类完全不同。解决方法是，如果继承没有意义并且子类和父类之间确实没有共同点，可以消除继承。

5） 冗类。

即一个做的事情太少的类。你所创建的每一个类都得有人理解和维护，这些工作都是要花费成本的。这类出现的主要原因如下:①某个类因为重构不再做那么多的工作；②开发者

事前规划了某些变化并添加一个类来应付这些变化，但变化实际上没有发生。

如果某些子类没有做足够的工作，尝试使用崩溃层次结构（collapse hierarchy）对于几乎没用的组件可使用内联类。

6） 数据泥团。

当一些变量在程序的不同部分一起传播时会发生数据泥团现象。也就是说，常常可以在很多地方看到相同的数据:两个类中相同的字段、许多函数签名中相同的参数。这些总是绑在一起出现的数据应该拥有属于它们自己的对象。

数据泥团的处理策略是——首先找出这些数据以字段形式出现的地方，运用 Extract Class 将它们提炼到一个独立对象中。然后将注意力转移到函数签名上，通过引入参数对象或保留整个对象为它“减肥”。这么做的直接好处是可以将很多参数列缩短，简化函数调用。

7） 不完美的程序库类。

尽管许多编程技术都建立在程序库类的基础上，但是程序库类构筑者没有未卜先知的能力，不可能考虑到所有可能的情况。麻烦的是库的形式往往不够好，不可能让我们修改其中的类以使它完成我们希望完成的工作。

我们有两个专门应付这种情况的工具。如果你只想修改程序库类内的一两个方法，可以通过引入外援方法（ Introduce Foreign Method）来实现；如果想要添加一大堆额外行为，就得通过引入局部扩展机制（Introduce Local Extension）来实现。

8） 幼稚的数据类。

所谓数据类是指它们拥有一些字段，以及用于访问这些字段的方法，除此之外一无长物。这样的类只是一种“不会说话的数据容器”，它们几乎一定被其他类过分细地操控着。这些类早期可能拥有 public 字段，果真如此你应该在别人注意到它们之前，立刻运用 Encapsulate Field 将它们封装起来。如果这些类内含容器类的字段，你应该检查它们是不是得到了恰当的封装；如果没有，就运用 Encapsulate Collection 把它们封装起来。对于那些不该被其他类修改的字段，请使用移走设置方法（ Remove Setting Method）。

1.3 方法级坏味道

过长的方法、方法参数太多、超长标识符、超短标识符、数据过量返回和超长代码行等，都隶属于方法级代码坏味道。

过长方法即一个方法（或者函数、过程）中含有太多行代码。一般来说，对于任何超过 10 行的方法，你就可以考虑其是不是过长了原则上函数中的代码行数不要超过 100 行。

大部分人都觉得:“我就添加这么两行代码，为此新建一个方法实在是小题大做了。”于是，张三加两行，李四加两行，王五加两行，方法日益庞大，最终再也无人能完全看懂了。于是大家就更不敢轻易动这个方法了，只能恶性循环地往其中添加代码。所以，如果你看到一个超过 200 行的方法，通常它都是多个程序员东拼西凑出来的。

```
employ.java
1
2
3  public class employ {
4
5      public static void main(String[] args) {
6      String data="ruchi dong_30.00_40_2000.0,200.0,600.0";
7      employee empone=new employee(data);
8   try{
9          if(employee.getearnings()==1256.0)System.out.println(employee.getname()+"   "+"pass"+employee.getearnings()+"");
10       else System.out.println(employee.getname()+"   "+"not pass"+employee.getearnings());
11  }catch (NumberFormatException iae){System.out.println("failed"+iae);}
12     String data2="lao wang_80.00_60_6000.0,250.0,700.0,800.0,900.0";
13     employee empone2=new employee(data2);
14  try{
15         if(employee.getearnings()==4973.0)System.out.println(employee.getname()+"   "+"pass"+employee.getearnings()+"");
16       else System.out.println(employee.getname()+"   "+"not pass"+employee.getearnings());
17  }catch (NumberFormatException iae){System.out.println("failed"+iae);}
18     String data3="zhangsan_40.00_30_4000.0,2000.0,3600.0,4530.0";
19     employee empone3=new employee(data3);
20  try{
21         if(employee.getearnings()==1482.6)System.out.println(employee.getname()+"   "+"pass"+employee.getearnings()+"");
22       else System.out.println(employee.getname()+"   "+"not pass"+employee.getearnings());
23  }catch (NumberFormatException iae){System.out.println("failed"+iae);}
24
25
26
27
28
29     }
30 }
31
```

样例：充满坏味道的代码

备注：
可以通过代码重构、复用设计模式、学习和运用编码规范来降低或消除代码坏味道。

第 2 节 程序静态分析工具 SourceMonitor

符合代码风格只是第一步，漂亮的代码还要易于理解。让人容易理解的话，单个函数的代码行就不能太长，嵌套层数就不能太多，分支条件判断不能太多。这些工作不是代码风格能解决的。我们需要代码度量工具-SourceMonitor。

SourceMonitor 能够检查文件中函数的个数，每个函数的代码行数，注释比例，函数的调用深度，圈复杂度等。 其中需要最关键的是每个函数圈复杂度和每个函数的代码行数。圈复杂度是指函数中可独立执行的路径，因此每出现一次 if/else/while,switch/case/break 等,圈复杂度就加 1.圈复杂度越高，说明函数中可执行的路径越多，也就越复杂。超过一定值如(15 或者 10)以后就要考虑能否将函数重构了。此外一个函数的代码行数如果太长，不能一屏显示的话，不容易让人记住和理解，也需要对该函数进行提炼。

2.1 安装及配置

SourceMonitor 是一款免费的软件，运行在 Windows 平台下。它可对多种语言写就的代码进行度量，包括 C、C++、C#、Java、VB、Delphi 和 HTML，并且针对不同的语言，输出不同的代码度量值。它只关注代码度量，并为程序员提供及时的反馈，为代码重构提供启发式信息。

（1） 安装向导

访问 http://www.campwoodsw.com/ 下载 SourceMonitorV3.5.8.15，安装包为 SMSetupV358.exe。然后，双击，完成安装。

例如，把 SourceMonitor 安装在 C 盘根下，安装目录名称为 C:\SourceMonitor，安装目录结构如下图 1 示：

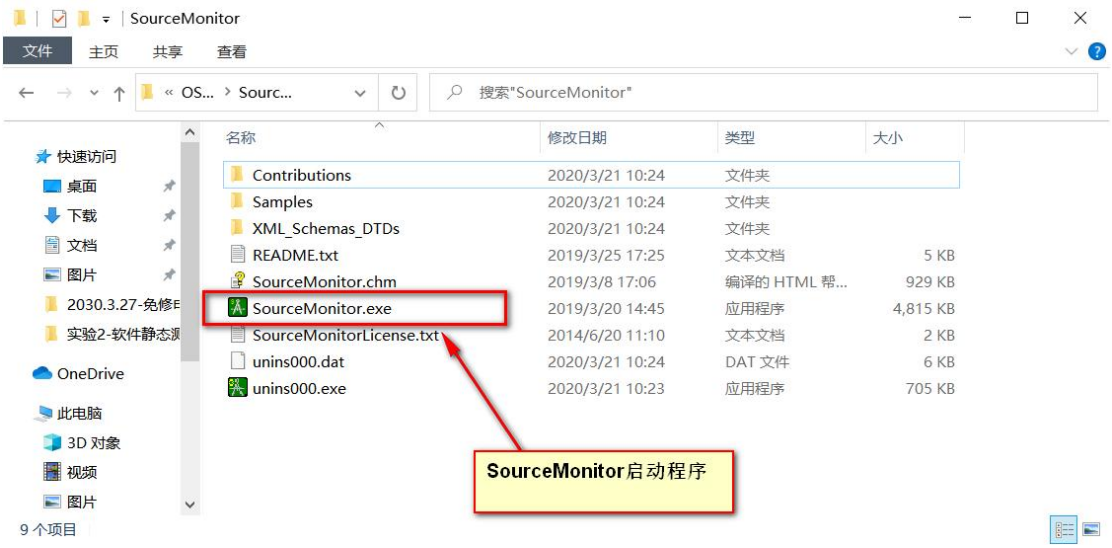


图 1 SourceMonitor 安装目录结构

（2） 使用模式

① 单机模式

双击 SourceMonitor 桌面图标/SourceMonitor 启动程序，运行该软件。

② 嵌套模式

把 SourceMonitor 作为 Eclipse、IDEA 等软件集成开发环境的外挂插件，通过 Eclipse、IDEA 等软件的菜单栏调用并启动 SourceMonitor。

以 Eclipse 中设置 SourceMonitor 外挂插件为例。

步骤 1：点击菜单项“Run→External Tools→External Tools Configuration”，打开“External Tools Configurations”对话框。

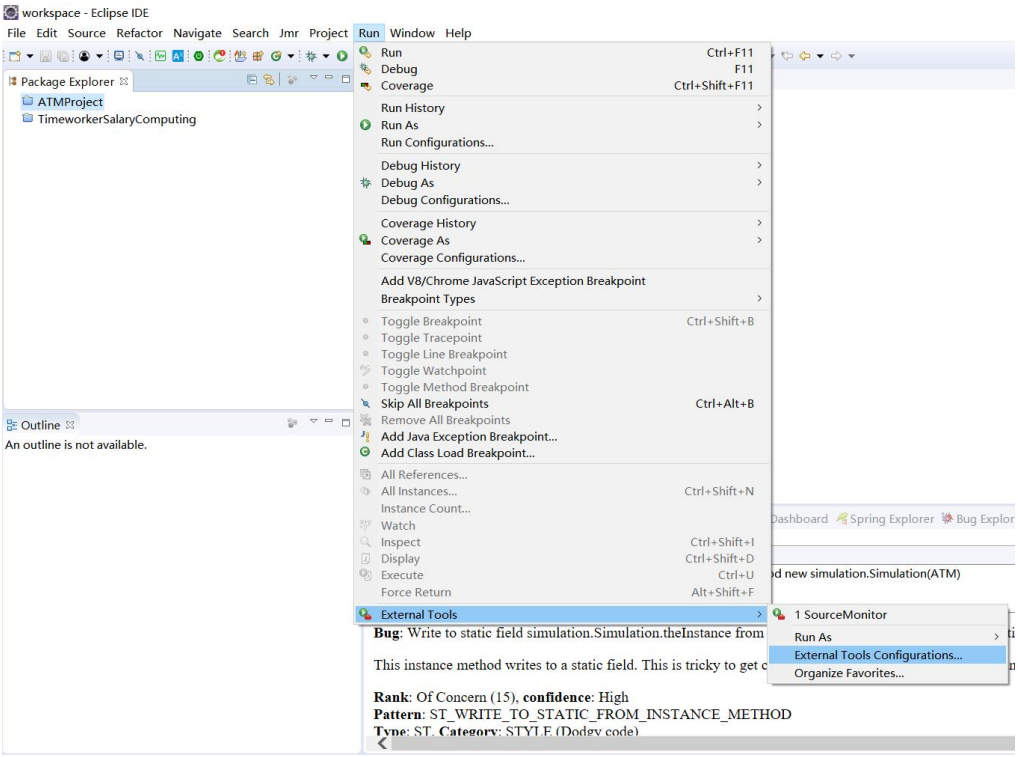


图 2 “External Tools Configurations”菜单项

步骤 2：在“External Tools Configurations”对话框中，鼠标右键点击导航栏中的“Program”右键打开“New Configuraion”配置页，新建一个 Program 对象。

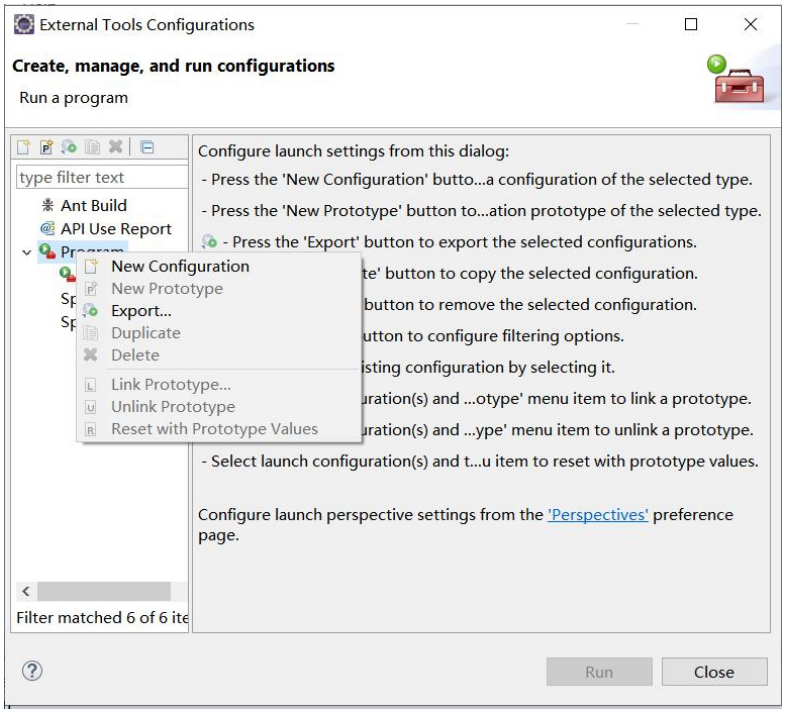


图 3 建立新的外部应用

步骤 3：设置 Program 参数：Name= SourceMonitor；点击“Browse file system”，选择 SourceMonitor 启动程序所在位置为“C:\SourceMonitor\SourceMonitor.exe”；SourceMonitor 启动参数设为“/DJava ${container_loc}/${resource_name}”。

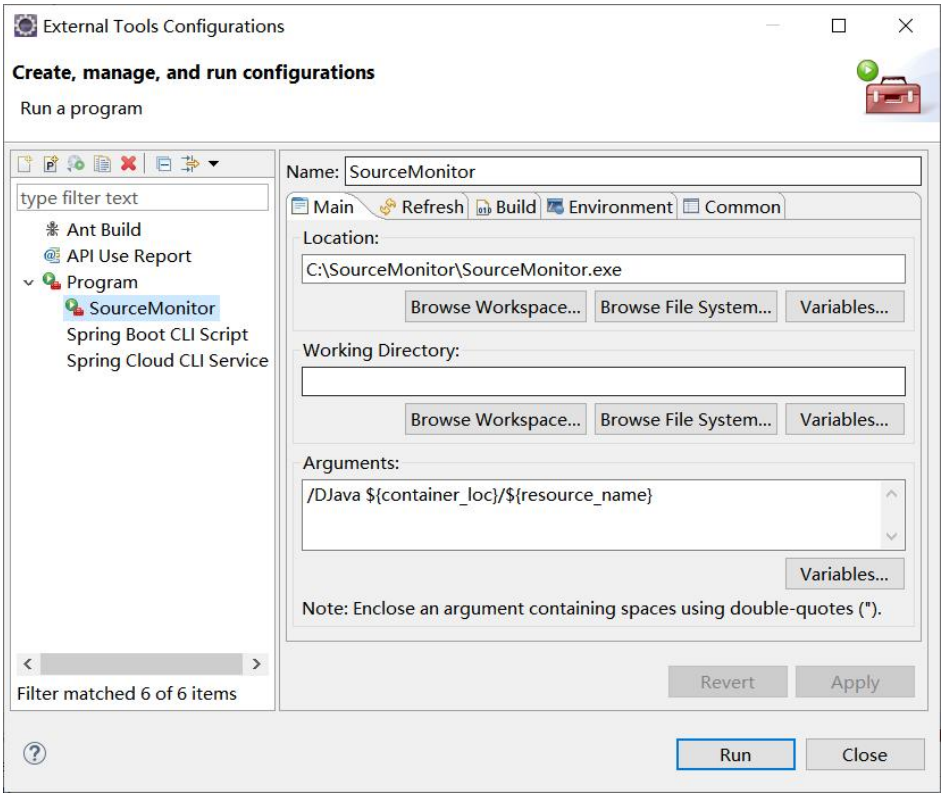


图 4 外部插件选项卡设置

步骤 4：在 Eclipse 项目中，鼠标焦点放在 Java 源程序上，然后选择“Run→External

Tools→External Tools Configuration→SourceMonitor”调用 SourceMonitor，出现 Java 程序的度量结果。

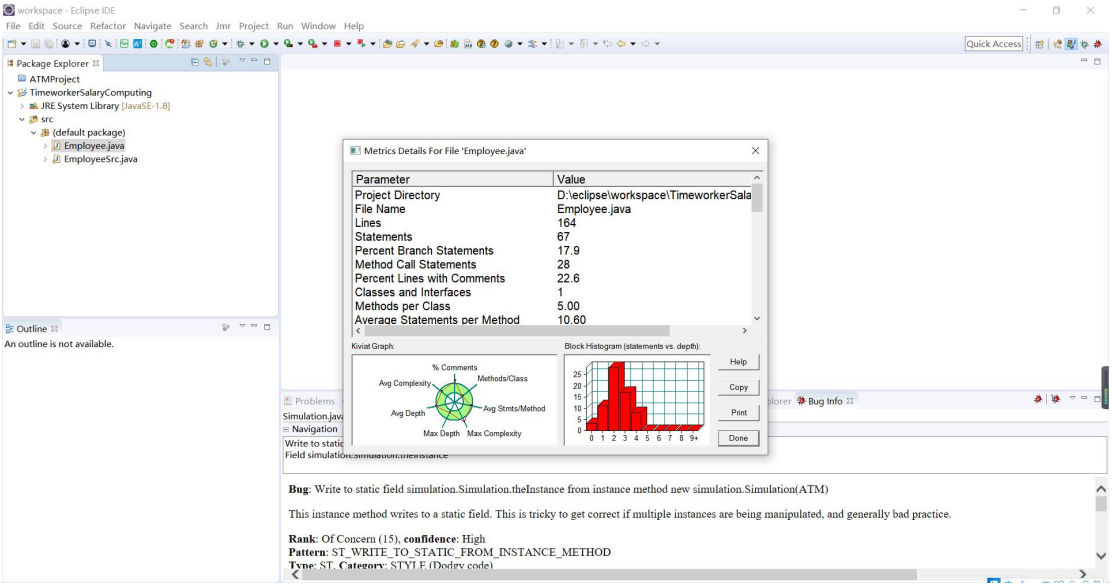


图 5 通过 Eclipse 调用 SourceMonitor

2.2 程序度量指标解读

（1）代码行数 Lines

包括空行在内的代码行数。

（2）语句数（Statements）

在 C++中，语句是以分号结尾的。分支语句 if，循环语句 for、while，跳转语句 goto 都被计算在内，预处理语句#include、#define 和#undef 也被计算在内。

（3）分支语句比例（Percent Branch Statements）

该值表示分支语句占语句数目的比例，包括 if、else、for、while、break、continue、goto、switch、case、default 和 return。另外，异常处理的 catch 也被作为 1 个分支计算。

（4）注释比例（Percent Lines with Comments）

该值是指注释行（包括/*……*/和//……情势的注释）与总代码行数的比例。

（5）类个数（Classes）

包括 class、struct 和 template 在内的个数。

（6）平均每一个类方法数（Methods per Class）

源码项目中，所有类拥有的方法数/类的个数。

（7）函数个数（Functions），即程序中的函数个数。

（8）函数语句数目的平均值（Average Statements per Method）

总的函数语句数目除以函数数目得到该值。

（9）函数圈复杂度（Function Complexity）

圈复杂度是指函数中可履行路径的数目，以下语句为圈复杂度的值贡献 1：if/else/for/while 语句，3 元运算符语句，if/for/while 判断条件中的"&&"或“||”，switch 语句，后接 break/goto/return/throw/continue 语句的 case 语句，catch/except 语句等。对应有最大圈复杂度（Max Complexity）和平均圈复杂度（Avg Complexity）。

（10）函数深度（Block Depth）

函数深度是指函数中分支嵌套的层数。对应有最大深度（Max Depth）和平均深度（Avg

Depth）。

（11）软件模块接口相关的度量指标

软件模块接口相关的度量指标包括模块的扇入数、扇出数。此外，将从软件架构层次上，把多个模块及其关系表示为层次方框图，然后衡量软件架构的深度、宽度。从经验上看，一个模块的扇出、扇入数控制在 7 以内，以避免复杂且难以维护的软件模块。

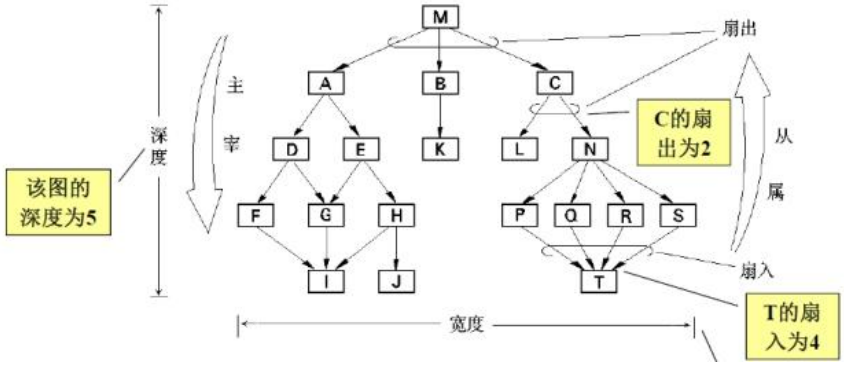


图 6 软件模块接口相关的度量指标

2.3 如何计算函数的圈复杂度

例：

```
1  public String case2(int index, String string) {
2      String returnString = null;
3      if (index < 0) {
4          throw new IndexOutOfBoundsException("exception <0 ");
5      }
6
7      if (index == 1) {
8          if (string.length() < 2) {
9              return string;
10         }
11
12         returnString = "returnString1";
13     } else if (index == 2) {
14         if (string.length() < 5) {
15             return string;
16         }
17
18         returnString = "returnString2";
19     } else {
20         throw new IndexOutOfBoundsException("exception >2 ");
21     }
22
23     return returnString;
24 }
```

步骤 1：把源码转换程流程图

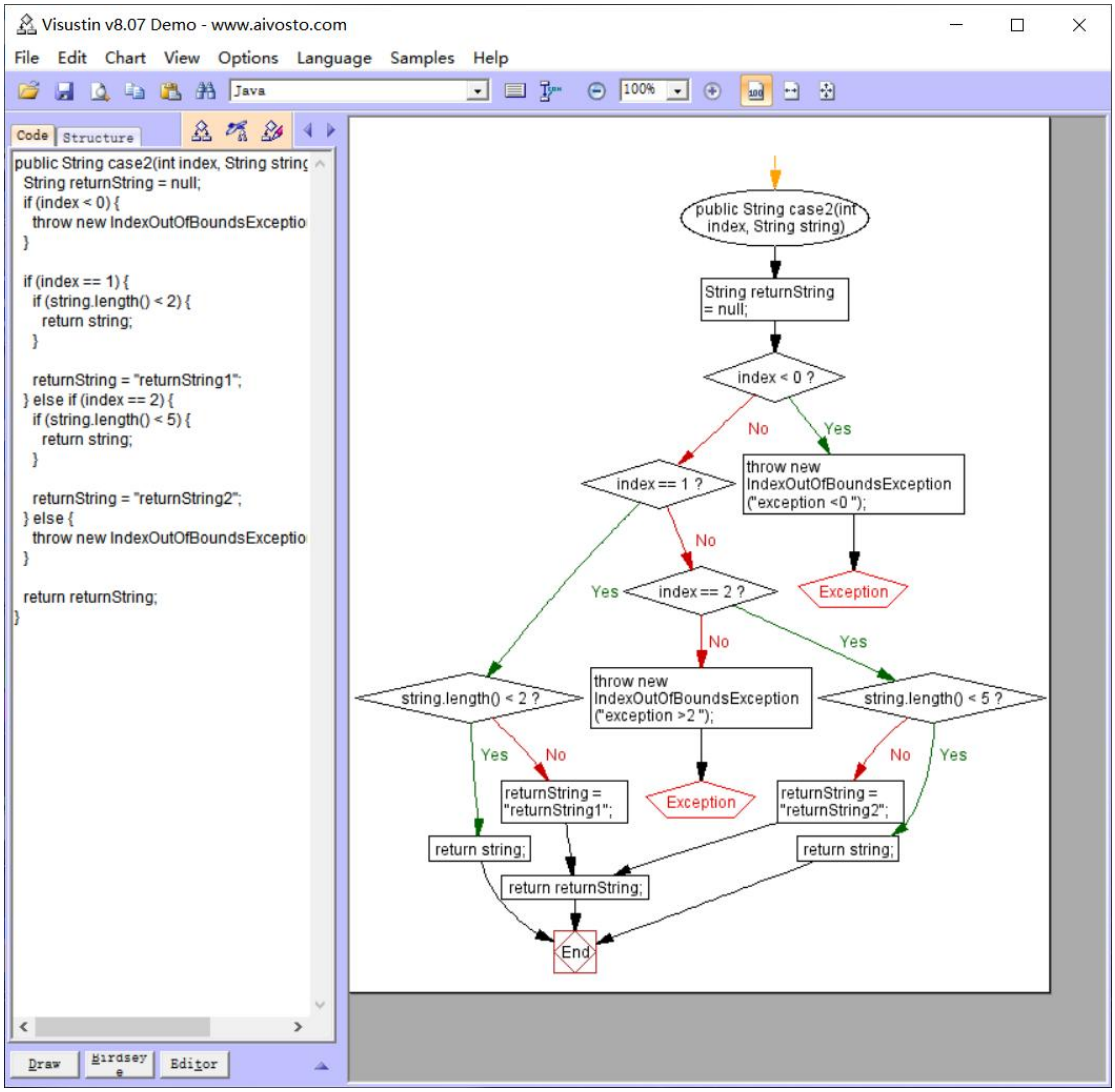
源码-->流程图转换，可以人工方式进行，也可以借助工具（例如）从源码生成流程图。

借助 visustin v8 把 Java 源码转换为流程图原型，再人工进行流程图润色，添加表示控制流流向的控制流，确保程序是单入口和单出口的。

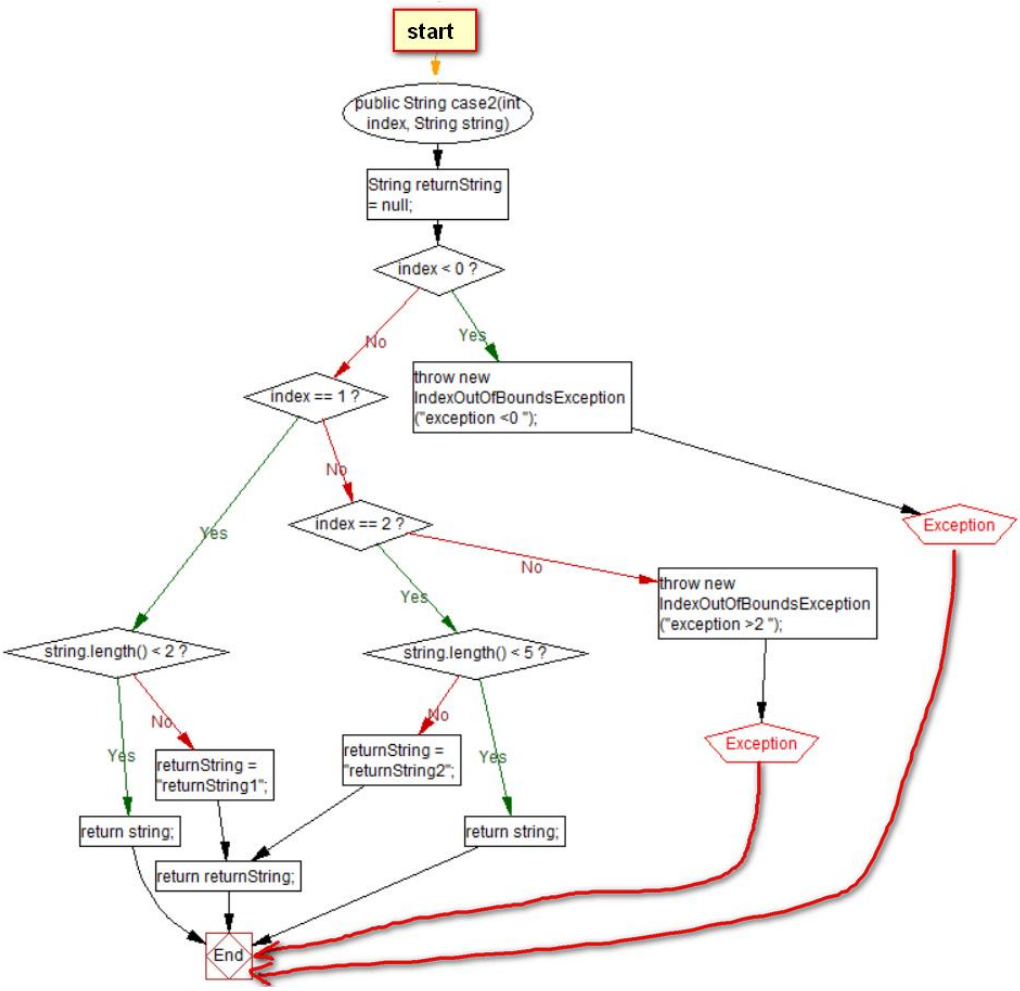
Tips：

Visustin v8 is a language and compatibility update for all existing users. New languages include AutoIt, GW-BASIC, PowerBASIC and RPG. Create tag charts from HTML and XML. Visustin v8 updates support for 21 languages, including modern JavaScript and the languages of Visual Studio 2015. Visustin v8 is compatible with Windows 10, Word 2016 and Visio 2016, and earlier. Whether you're working with modern or legacy code, Visustin v8 is a must for you.

从官网 https://www.aivosto.com/shareware/visus807.zip 下载 visustin v8 Demo 演示版（30 天试用）。安装后将 Java 源码复制到 Visustin 的 code 选项卡内，点击按钮“Draw”将 Java 源码转换为流程图。



使用 FastStone Capture 截屏，添加流程图的辅助控制流连接线，使得流程图为单入口、单出口的结构。



步骤 2：将流程图转换为流图

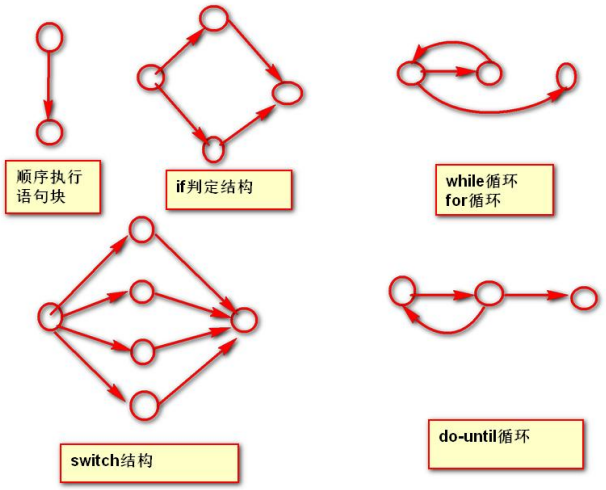
首先,在 FastStone Capture 中，对流程图中的语句进行归类，形成流图的初稿；

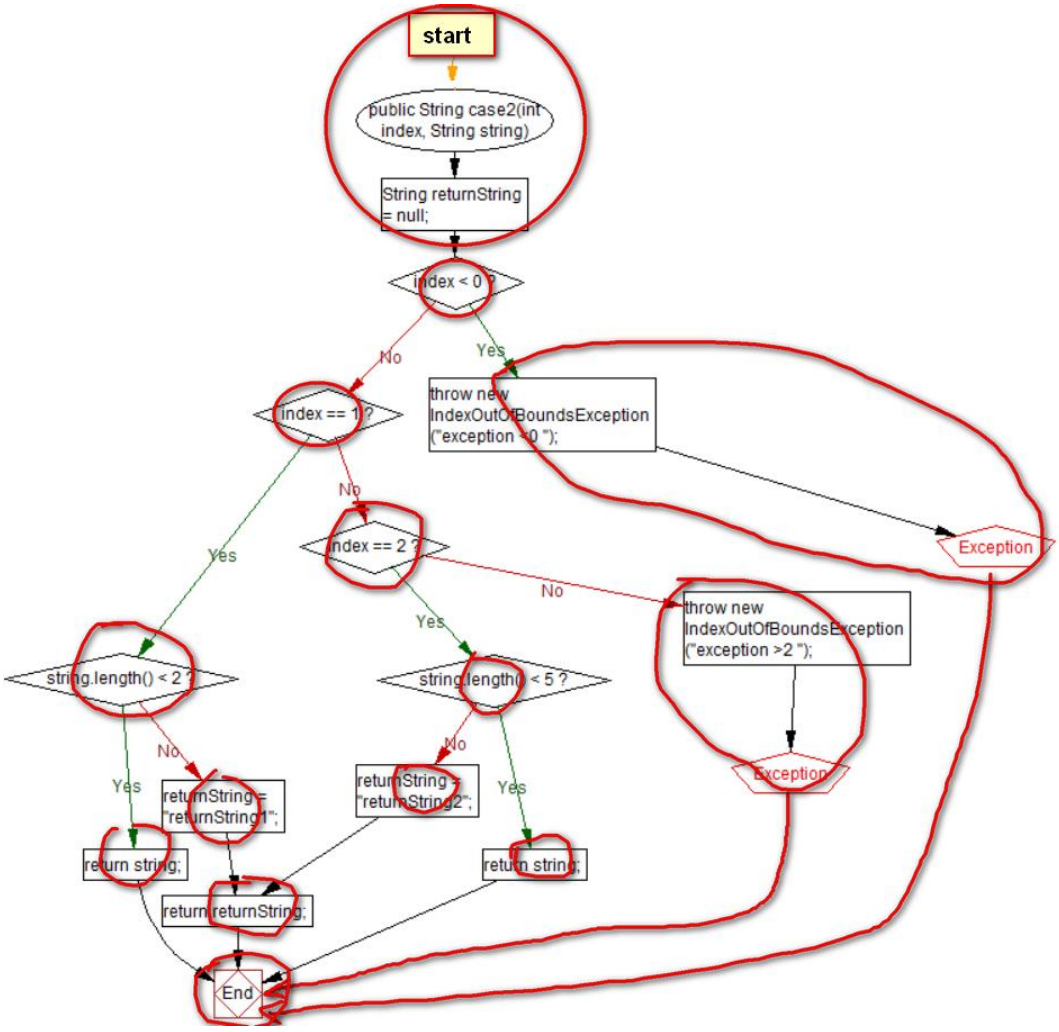
然后，用 Visio 绘制出流图定稿。

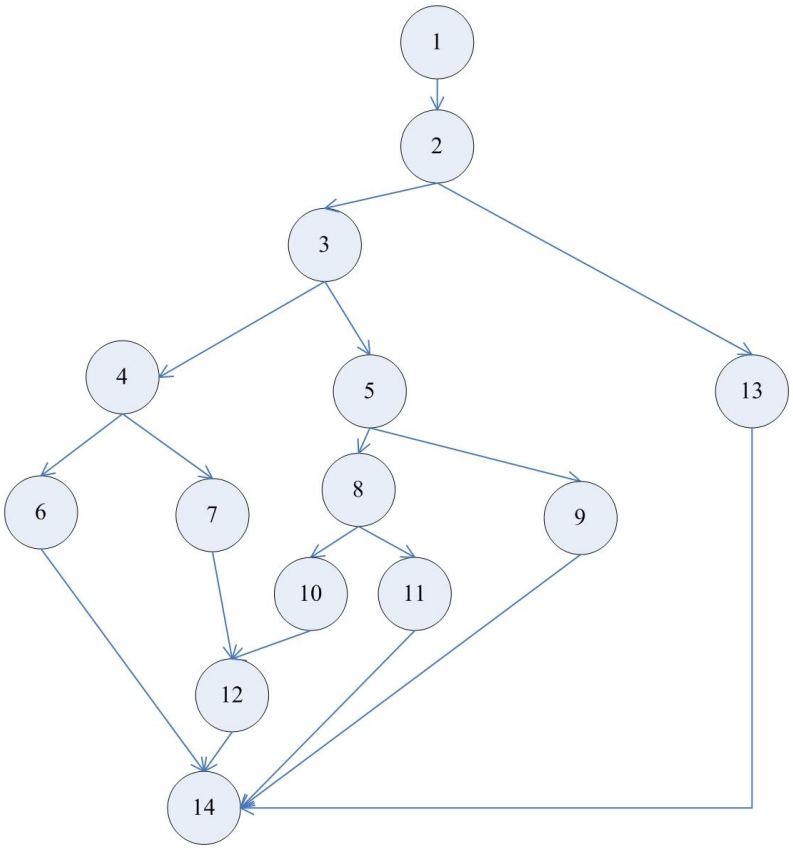
Tips:

流图由节点和边构成，描述程序的控制流结构。

流图中控制流模式包括：




start
public String case2(int index, String string)
String returnString = null;
index < 0 ?
No
Yes
throw new IndexOutOfBoundsException ("exception <0 ");
Exception
index == 1 ?
No
Yes
index == 2 ?
No
Yes
throw new IndexOutOfBoundsException ("exception >2 ");
string.length() < 2 ?
string.length() < 5 ?
Exception
No
No
Yes
Yes
returnString = "returnString1";
returnString = "returnString2";
return string;
return string;
return returnString;
End



Visio 绘制，形成流图的定稿。

步骤 3：计算圈复杂度 V(G)值
V(G)计算方法有三种：
方法 1：V(G)=流图中的区域数=5+1=6
流图中，区域为流程边与边连接形成的封闭面积。备注，流图中最外层边与边构成一个开放的面积，记为“开放的区域”。即，流图中区域数=封闭区域数+开放区域数（固定为 1）=封闭区域数+1
方法 2：V(G)=边数量-节点数+2=18-14+2=6
方法 3：V(G)=判定节点数+1=5+1=6

Tips：
函数的圈复杂度控制在 10 以内。圈复杂度高时，必须重构代码，以降低圈复杂度。

第 3 节 Java 编码规范

现代软件架构的复杂性需要协同开发完成，如何高效地协同呢？无规矩不成方圆，无规范难以协同。对软件来说，适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。代码的字里行间流淌的是软件系统的血液，质量的提升是尽可能少踩坑，杜绝踩重复的坑，切实提升系统稳定性，码出质量。

本节从 Java 开发者视角，参照《阿里巴巴 Java 开发手册 1.4.0》讲述 Java/JavaEE 编码规范及注意事项。

3.1 命名规范

规则 1-1：代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

规则 1-2：命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。

规则 1-3：类名使用 UpperCamelCase 风格

规则 1-4：方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

规则 1-5：常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。例如，MAX_STOCK_COUNT

规则 1-6：抽象类命名使用 Abstract 或 Base 开头；异常类命名使用 Exception 结尾；测试类命名以它要测试的类的名称开始，以 Test 结尾。

规则 1-7：类型与中括号紧挨相连来 表示 数组。例如，定义整形数组 int[] arrayDemo;

规则 1-8：包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式。

规则 1-9：杜绝完全不规范的缩写，避免望文不知义。

规则 1-10：为了达到代码自解释的目标，任何自定义编程元素在命名时，使用尽量完整的单词组合来表达其意。

规则 1-11：如果模块、接口、类、方法使用了设计模式，在命名时需体现出具体模式。将设计模式体现在名字中，有利于阅读者快速理解架构设计理念。例如，public class LoginProxy; public class ResourceObserver;

规则 1-12：接口和实现类的命名有两套规则：
1）对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类用 Impl 的后缀与接口区别。 例如，CacheServiceImpl 实现 CacheService 接口。
2） 如果是形容能力的接口名称，取对应的形容词为接口名（通常是－able 的形式）。例如，AbstractTranslator 实现 Translatable 接口。

3.2 常量定义

规则 2-1：不允许任何魔法值（即未经预先定义的常量）直接出现在代码中。

规则 2-2：在 long 或者 Long 赋值时，数值后使用大写的 L，不能是小写的 l，小写容易跟数

字 1 混淆，造成误解。

反例：Long a = 2l; 写的是数字的 21，还是 Long 型的 2?

规则 2-3：不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护。 说明：大而全的常量类，杂乱无章，使用查找功能才能定位到修改的常量，不利于理解和维护。

规则 2-4：常量的复用层次有五层：跨应用共享常量、应用内共享常量、子工程内共享常量、包内共享常量、类内共享常量。
1） 跨应用共享常量：放置在二方库中，通常是 client.jar 中的 constant 目录下。
2） 应用内共享常量：放置在一方库中，通常是子模块中的 constant 目录下。
3） 子工程内部共享常量：即在当前子工程的 constant 目录下。
4） 包内共享常量：即在当前包下单独的 constant 目录下。
5） 类内共享常量：直接在类内部 private static final 定义。

规则 2-5：如果变量值仅在一个固定范围内变化用 enum 类型来定义。 说明：如果存在名称之外的延伸属性应使用 enum 类型，下面正例中的数字就是延伸信息，表示一年中的第几个季节。

正例：

```
public enum SeasonEnum {
    SPRING(1), SUMMER(2), AUTUMN(3), WINTER(4);
    private int seq;
    SeasonEnum(int seq){
        this.seq = seq;
    }
}
```

3.3 代码格式

规则 3-1：合理使用换行
如果是大括号内为空，则简洁地写成{}即可，不需要换行；
如果是非空代码块则： 1） 左大括号前不换行；2） 左大括号后换行； 3）右大括号前换行；4）右大括号后还有 else 等代码则不换行；5）表示终止的右大括号后必须换行。

规则 3-2：左小括号和字符之间不出现空格；同样，右小括号和字符之间也不出现空格。
规则3-3：if/for/while/switch/do等保留字与括号之间都必须加空格。
规则3-4：任何二元、三元运算符的左右两边都需要加一个空格。
规则3-5：注释的双斜线与注释内容之间有且仅有一个空格。

规则3-6：单行字符数限制不超过 120 个，超出需要换行，换行时 遵循如下原则：
1）第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进，参考示例。
2）运算符与下文一起换行。
3）方法调用的点符号与下文一起换行。
4）方法调用中的多个参数需要换行时，在逗号后进行。
5）在括号前不要换行。

规则 3-7：方法参数在定义和传入时，多个参数逗号后边必须加空格。
规则 3-8：【推荐】单个方法的总行数不超过 80 行。
规则 3-9：代码逻辑分清红花和绿叶，个性和共性，绿叶逻辑单独出来成为额外方法，使主干代码更加清晰；共性逻辑抽取成为共性方法，便于复用和维护。
规则 3-10：不同逻辑、不同语义、不同业务的代码之间插入一空行分隔开，提升可读性。
例 1：

```
public static void main(String[] args) {

    // 缩进 4 个空格
    String say = "hello";
    // 运算符的左右必须有一个空格
    int flag = 0;
    // 关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号，0 与右括号不需要空格
    if (flag == 0) {
        System.out.println(say);
    }

    // 左大括号前加空格且不换行；左大括号后换行
    if (flag == 1) {
        System.out.println("world");
    // 右大括号前换行，右大括号后有 else，不用换行
    } else {
        System.out.println("ok");
    // 在右大括号后直接结束，则必须换行
    }
}
```

例 2：

正例：

```
    StringBuffer sb = new StringBuffer();
    // 超过 120 个字符的情况下，换行缩进 4 个空格，点号和方法名称一起换行
    sb.append("zi").append("xin")...
        .append("huang")...
        .append("huang")...
        .append("huang");
```

反例：

```
    StringBuffer sb = new StringBuffer();
    // 超过 120 个字符的情况下，不要在括号前换行
    sb.append("zi").append("xin")...append
        ("huang");

    // 参数很多的方法调用可能超过 120 个字符，不要在逗号前换行
     method(args1, args2, args3, ...
        , argsX);
```

3.4 OOP（面向对象编程）规约

规则 4-1：避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。

规则 4-2：所有的覆写方法，必须加@Override 注解。

规则 4-3：外部正在调用或者二方库依赖的接口，不允许修改方法签名，避免对接口调用方产生影响。接口过时必须加@Deprecated 注解，并清晰地说明采用的新接口或者新服务是什么。

规则 4-4：不能使用过时的类或方法。
例如，java.net.URLDecoder 中的方法 decode(String encodeStr) 这个方法已经过时，应该使用双参数 decode(String source, String encode)。接口提供方既然明确是过时接口，那么有义务同时提供新的接口；作为调用方来说，有义务去考证过时方法的新实现是什么。

规则 4-5：所有的相同类型的包装类对象之间值的比较，全部使用 equals 方法比较。

规则 4-6：序列化类新增属性时，请不要修改 serialVersionUID 字段，避免反序列失败；如果完全不兼容升级，避免反序列化混乱，那么请修改 serialVersionUID 值。

规则 4-7：构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。

规则 4-8：类内方法定义的顺序依次是：公有方法或保护方法>私有方法> getter/setter 方法。

规则4-9：循环体内，字符串的连接方式，使用StringBuilder的append方法进行扩展。
例如反编译出的字节码文件显示每次循环都会new出一个StringBuilder对象，然后进行append操作，最后通过toString方法返回String对象，造成内存资源浪费。

反例：

```
String str = "start";
for (int i = 0; i < 100; i++) {
    str = str + "hello";
}
```

规则 4-10：final 可以声明类、成员变量、方法、以及本地变量，下列情况使用 final 关键字：
1） 不允许被继承的类，如：String 类。
2） 不允许修改引用的域对象。
3） 不允许被重写的方法，如：POJO 类的 setter 方法。
4） 不允许运行过程中重新赋值的局部变量。
5） 避免上下文重复使用一个变量，使用 final 描述可以强制重新定义一个变量，方便更好地进行重构。

规则 4-11：类成员与方法访问控制从严：
1） 如果不允许外部直接通过 new 来创建对象，那么构造方法必须是 private。
2） 工具类不允许有 public 或 default 构造方法。
3） 类非 static 成员变量并且与子类共享，必须是 protected。
4） 类非 static 成员变量并且仅在本类使用，必须是 private。
5） 类 static 成员变量如果仅在本类使用，必须是 private。

6） 若是 static 成员变量，考虑是否为 final。
7） 类成员方法只供类内部调用，必须是 private。
8） 类成员方法只对继承类公开，那么限制为 protected。

规则4-12：任何数据结构的构造或初始化，都应指定大小，避免数据结构无限增长吃光内存。

规则 4-13：不要在视图模板中加入任何复杂的逻辑。 说明：根据 MVC 理论，视图的职责是展示，不要抢模型和控制器的活。

规则 4-14：在使用正则表达式时，利用好其预编译功能，可以有效加快正则匹配速度。

规则 4-15：注释掉不再使用的代码，务必同时给出必要的注释予以说明。
及时清理不再使用的代码段或配置信息。对于垃圾代码或过时配置，坚决清理干净，避免程序过度臃肿，代码冗余。

3.5 控制语句

规则 5-1：在一个 switch 块内，每个 case 要么通过 break/return 等来终止，要么注释说明程序将继续执行到哪一个 case 为止；在一个 switch 块内，都必须包含一个 default 语句并且放在最后，即使空代码。

规则 5-2：在 if/else/for/while/do 语句中必须使用大括号。即使只有一行代码，避免采用单行的编码方式：if (condition) statements;

规则 5-3：在高并发场景中，避免使用 等于 判断作为中断或退出的条件。

规则 5-4：除常用方法（如 getXxx/isXxx）等外，不要在条件判断中执行其它复杂的语句，将复杂逻辑判断的结果赋值给一个有意义的布尔变量名，以提高可读性。

规则 5-5：循环体中的语句要考量性能，以下操作尽量移至循环体外处理，如定义对象、变量、获取数据库连接，进行不必要的 try-catch 操作（这个 try-catch 是否可以移至循环体外）。

规则 5-6：避免采用取反逻辑运算符。
说明： 取反逻辑不利于快速理解，并且取反逻辑写法必然存在对应的正向逻辑写法。
正例：使用 `if (x < 628)` 来表达 x 小于 628。
反例：使用 `if (!(x >= 628))` 来表达 x 小于 628。

规则 5-7：接口入参保护，这种场景常见的是用作批量操作的接口。

规则 5-8：下列情形，需要进行参数校验：
1） 调用频次低的方法。
2） 执行时间开销很大的方法。此情形中，参数校验时间几乎可以忽略不计，但如果因为参数错误导致中间执行回退，或者错误，那得不偿失。
3） 需要极高稳定性和可用性的方法。

4） 对外提供的开放接口，不管是 RPC/API/HTTP 接口。
5） 敏感权限入口。

3.6 注释规范

规则 6-1：类、类属性、类方法的注释必须使用 Javadoc 规范，使用/**内容*/格式，不得使用// xxx 方式。

```
/**
* 获取用户信息
* @param  userId [用户id，对应数据库中的userid字段]
* <a href="/profile/547241" data-card-uid="547241" class="" target="_blank" data-card-index="3">
* @return    [返回单个用户或空]
*/
public User getUser(int userId )
{
    //TODO: 数据访问层返回id=userId的用户信息
}</a>
```

规则 6-2：所有的抽象方法（包括接口中的方法）必须要用 Javadoc 注释、除了返回值、参数、异常说明外，还必须指出该方法做什么事情，实现什么功能。

规则 6-3：所有的类都必须添加创建者和创建日期。

```
/**
* @Author Ryan
* @date 2020-03-28
*/
public class User{
}
```

规则 6-4：方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释使用/* */注释，注意与代码对齐。

```
/**
* [保存用户：保存至数据库中，并同步更新到缓存中]
* @param user [新用户信息：管理员新建用户，用户自主注册用户]
*/
public void addUser(User user){
  //保存用户到数据库中
  dao.addUser(user);
  /*将用户放置到缓存中
      key来自于user的userId加前缀 "CK_USER_"
          (缓存键_用户,定义于CacheKey定义的常量CK_USER上)
      value就是User本身
  */
  MapCache.getInstance().put(CacheKey.CK_USER, user);
}
```

规则 6-5：所有的枚举类型字段必须要有注释，说明每个数据项的用途。

```
/**
* 数据库类型
*/
public enum DbType
{
 /** Oracle,甲骨文数据库 */
 Oracle,
 /** Microsoft Sql Server Database,微软Sql Server数据库 */
 MsSql,
 /** MySql( my(the name of Monty's daughter ) sql),甲骨文收购的一个开源数据库 */
 MySql
}
```

规则 6-6：与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持英文原文即可。

规则 6-7：代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。 说明：代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后，就失去了导航的意义。

规则 6-8：对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作。

规则 6-9：好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。

反例：

```
// put elephant into fridge
put(elephant, fridge);
```

方法名 put，加上两个有意义的变量名 elephant 和 fridge，已经说明了这是在干什么，语义清晰的代码不需要额外的注释。

规则 6-10：特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。

1） 待办事宜（TODO）:（ 标记人，标记时间，[预计处理时间]） 表示需要实现，但目前还未实现的功能。这实际上是一个 Javadoc 的标签，目前的 Javadoc 还没有实现，但已经被广泛使用。只能应用于类，接口和方法（因为它是一个 Javadoc 标签）。

2） 错误，不能工作（FIXME）:（标记人，标记时间，[预计处理时间]） 在注释中用 FIXME 标记某代码是错误的，而且不能工作，需要及时纠正的情况。

```java
public class User
{
    /**
     * TODO:(ali,2020-04-01,2020-04-3)
     *     需要完成的内容：
     *         检查当前登录人是否有权限
     *         判断需要修改密码的用户的状态
     *         验证密码复杂度
     *
     * [重设密码，由用户自行输入密码]
     * @param userId [用户Id]
     * @param newPassword [用户输入的新密码]
     */
    public void resetPassword(int userId,String newPassword) {
    }

    /**
     * FIXME:(ali,2020-03-28,2020-03-31)
     * 错误描述：
     *     之前的删除用户操作是把用户直接从数据库删除了
     *     恢复用户是直接根据userId取用户，并进行操作
     *     因为被删除的用户getUser操作的是null，所以报异常了。
     * 修改方式：
     *     判断是否为null再进行修改。
     * 关联修改：
     *     将真删，变成标记。
     *     如果用户数量庞大，则分表操作，将被删用户转移到已删除的表中
     *         并将getUser方法区别对待。
     * [恢复用户]
     * @param userId [用户Id]
     */
    public void recoverUser(int userId){
    }
}
```

课程论文：

结合本节内容，观看《华山版<Java 开发手册>独家讲解》

https://developer.aliyun.com/live/1201?spm=a2c6h.14059151.1371164.1.55505738CtYqYU

1）详细介绍自己对阿里巴巴 Java 编码规范学习后的心得体会；编写《阿里巴巴<Java 开发手册>读后感》小论文，要求字数 1500 字左右；

2）结合自己在《面向对象程序设计（Java）》、专业实训时独立编码实现的 Java/JavaEE 源码项目，借鉴阿里巴巴 Java 编码规范，优化代码结构和编码风格。介绍自己如何对已有的 Java/JavaEE 源码项目进行代码优化的，描述代码优化过程及结果。

3）把文中涉及的 Java 源码项目打包提交，包括代码优化前后的源代码。

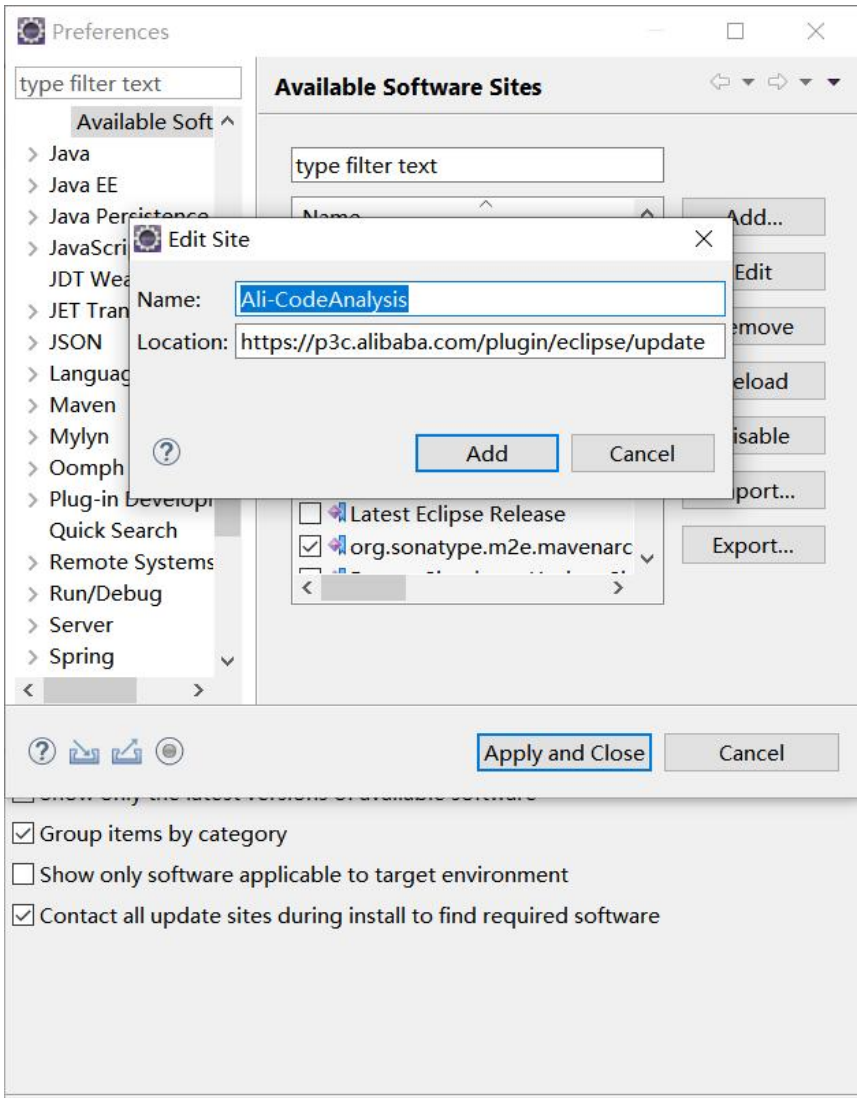
4）命名方式为“学号-姓名-Java 编码规范学习心得.zip”。提交截至日期 2022 年 3 月 23 日。提交到蓝墨云班课平台中。小论文提交到蓝墨云班课——课程考核：小论文一栏中。

第 4 节 程序静态分析工具 Ali-CodeAnalysis

4.1 插件安装

在 Eclipse 中，新建远程插件安装源，名称为 Ali-CodeAnalysis，地址为

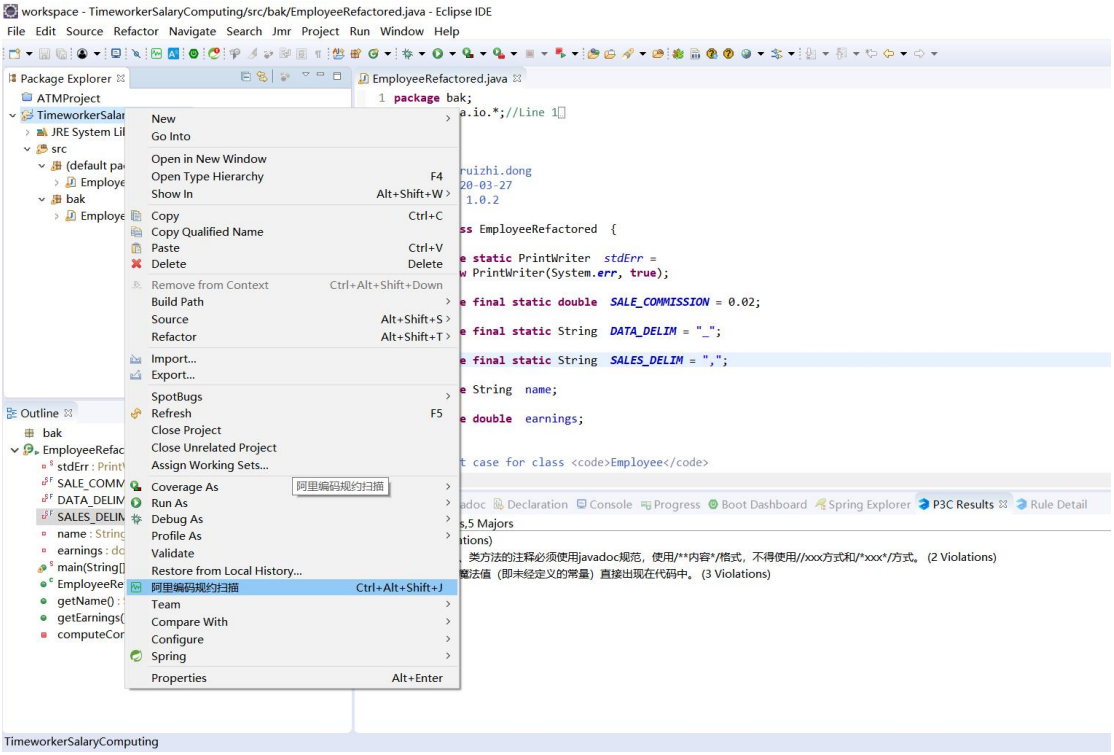
https://p3c.alibaba.com/plugin/eclipse/update



选择新建的安装源，完成 Ali-CodeAnalysis 安装。

4.2 程序静态分析

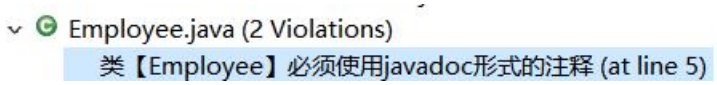
在 Eclipse 中，鼠标焦点放在要分析的程序或 Java 项目上，鼠标右键选择菜单项“阿里编程规约扫描”插件，进行 Java 程序扫描。



扫描结果显示在 P3C Results 视图之中。

4.3 代码重构

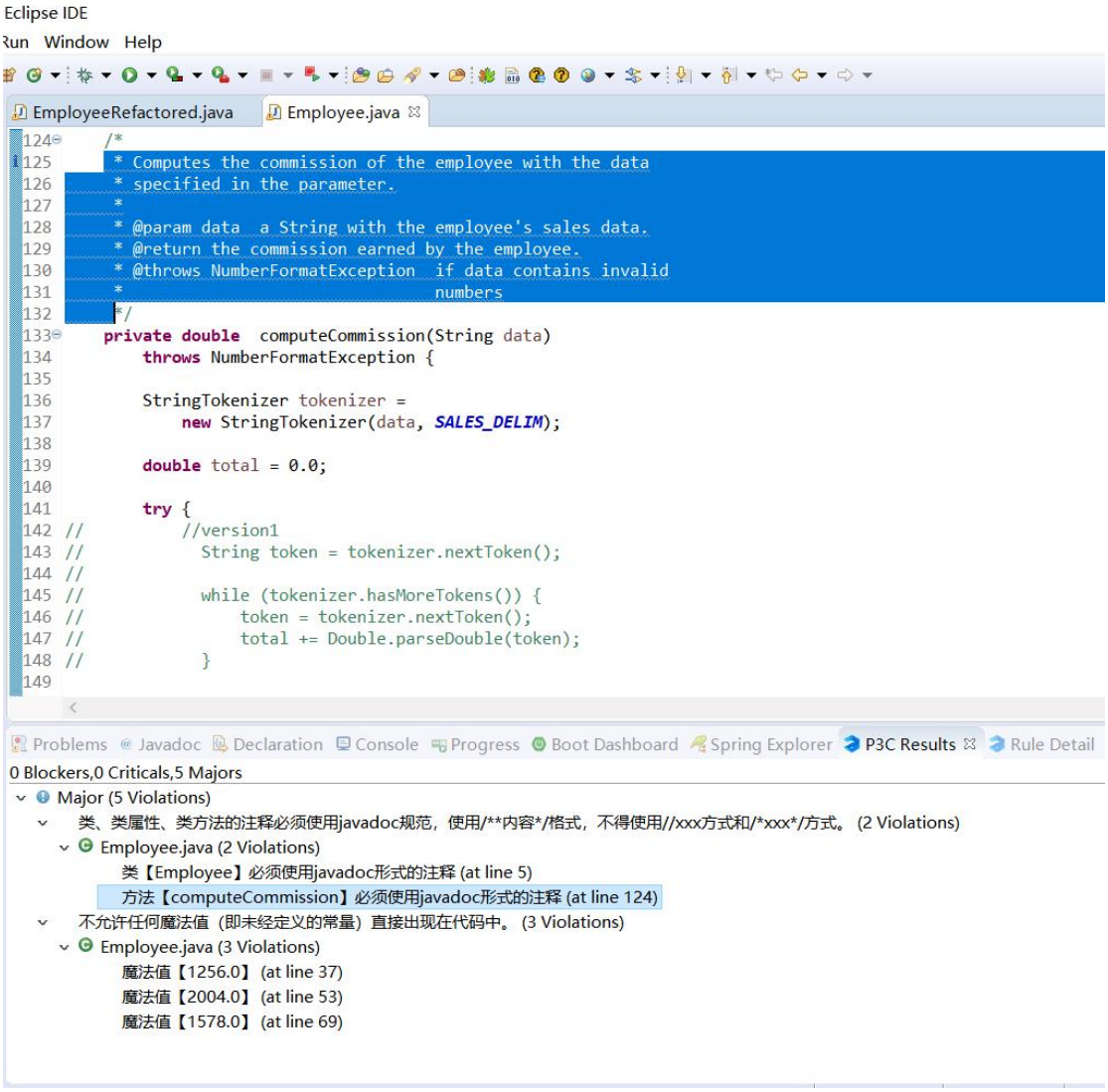
Bug1：



代码重构：

```
/**
* @Author ruizhi.dong
* @date 2020-03-27
* @version 1.0.2
*/
```

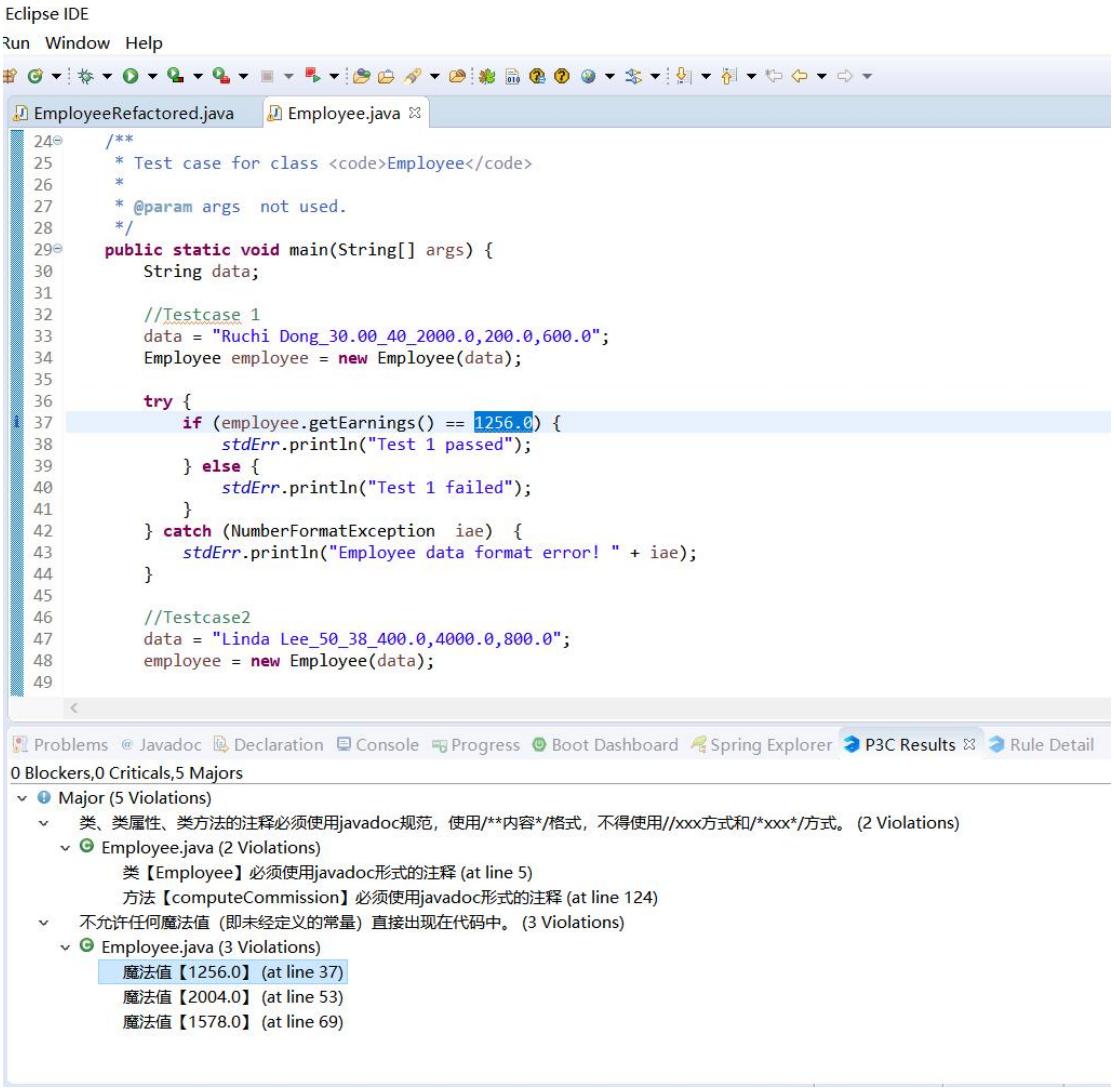
Bug2:



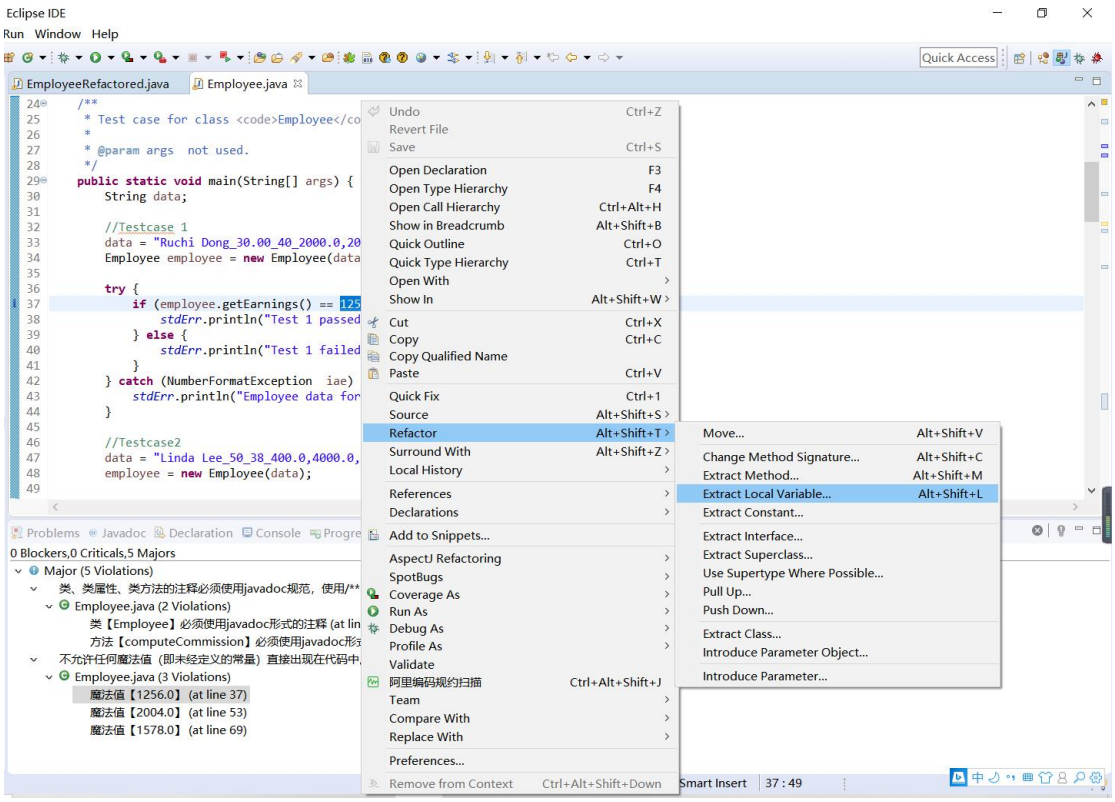
代码重构：

```
/**
 * [计算员工本周订单提成额]
 * @param data [员工本周销售金额流水]
 * @throws NumberFormatException [如果员工本周销售金额流水data字符串格式有错误]
 */
```

Bug3：魔法值[1256.0] (at line 37)


Eclipse IDE
Run Window Help
EmployeeRefactored.java
Employee.java
24 /**
25 * Test case for class <code>Employee</code>
26 *
27 * @param args not used.
28 */
29 public static void main(String[] args) {
30 String data;
31
32 //Testcase 1
33 data = "Ruchi Dong_30.00_40_2000.0,200.0,600.0";
34 Employee employee = new Employee(data);
35
36 try {
37 if (employee.getEarnings() == 1256.0) {
38 stdErr.println("Test 1 passed");
39 } else {
40 stdErr.println("Test 1 failed");
41 }
42 } catch (NumberFormatException iae) {
43 stdErr.println("Employee data format error! " + iae);
44 }
45
46 //Testcase2
47 data = "Linda Lee_50_38_400.0,4000.0,800.0";
48 employee = new Employee(data);
49
Problems Javadoc Declaration Console Progress Boot Dashboard Spring Explorer P3C Results Rule Detail
0 Blockers,0 Criticals,5 Majors
Major (5 Violations)
类、类属性、类方法的注释必须使用javadoc规范，使用/**内容*/格式，不得使用//xxx方式和/*xxx*/方式。 (2 Violations)
Employee.java (2 Violations)
类【Employee】必须使用javadoc形式的注释 (at line 5)
方法【computeCommission】必须使用javadoc形式的注释 (at line 124)
不允许任何魔法值（即未经定义的常量）直接出现在代码中。 (3 Violations)
Employee.java (3 Violations)
魔法值【1256.0】 (at line 37)
魔法值【2004.0】 (at line 53)
魔法值【1578.0】 (at line 69)

代码重构：



使用“Extract Local Variable”重构策略，新建一个局部变量 expectedResult。

```
36
37          try {
38              double expectedResultOfTestCase1 = 1256.0;
39
40              if (employee.getEarnings() == expectedResultOfTestCase1) {
41                  stdErr.println("Test 1 passed");
42              } else {
43                  stdErr.println("Test 1 failed");
44              }
45          } catch (NumberFormatException  iae)  {
46              stdErr.println("Employee data format error! " + iae);
47          }
```

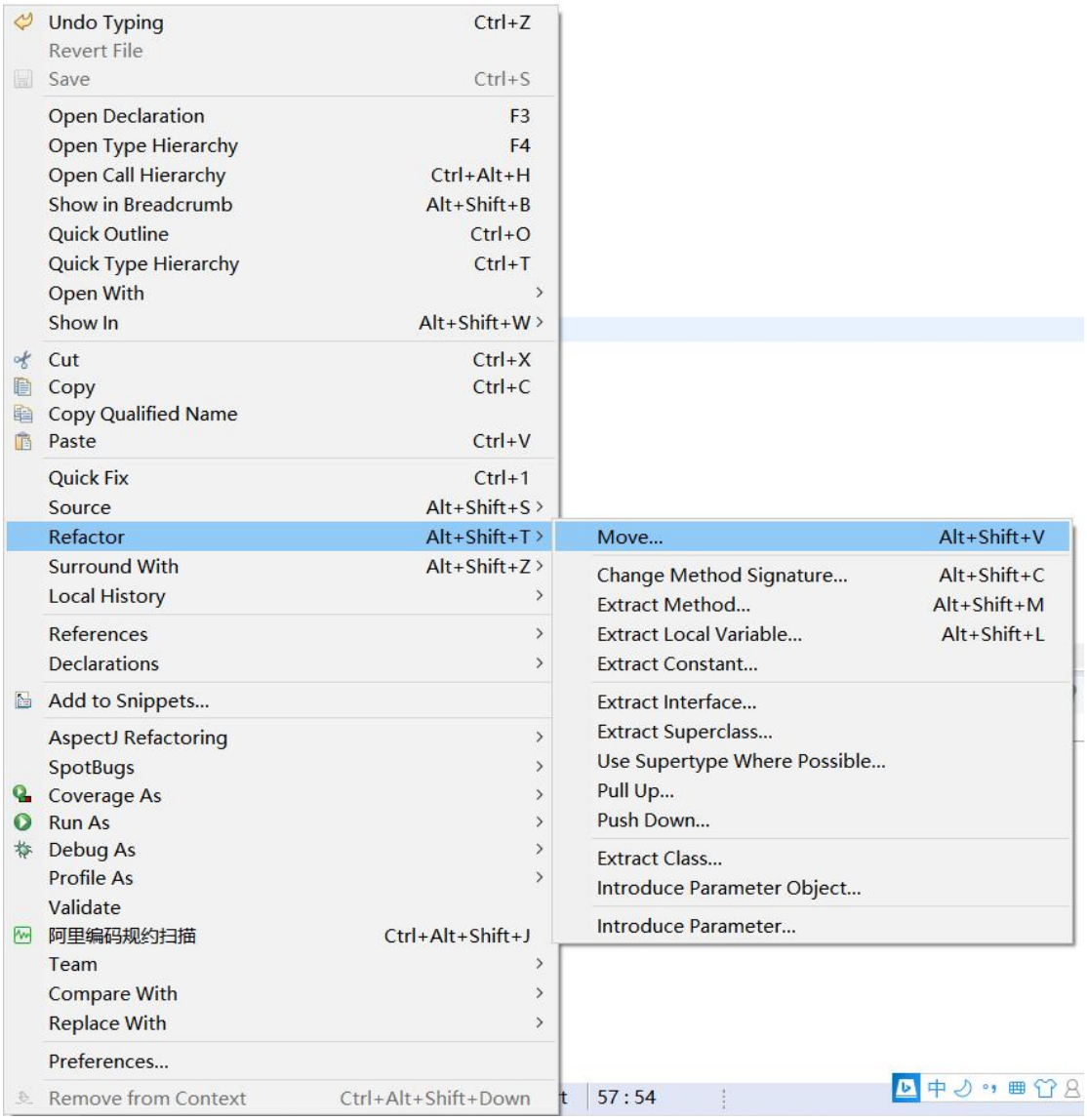
针对另外两个魔法值，采用“Extract Local Variable”进行代码重构结果如下：

Bug4：魔法值[2004.0] (at line 53)

Bug5：魔法值[1578.0] (at line 69)

```
50          //Testcase2
51          data = "Linda Lee_50_38_400.0,4000.0,800.0";
52          employee = new Employee(data);
53
54          String str="name";
55
56          try {
57              double expectedResultOfTestCase2 = 2004.0;
58              if (employee.getEarnings() == expectedResultOfTestCase2) {
59                  stdErr.println("Test 2 passed");
60              } else {
61                  stdErr.println("Test 2 failed");
62              }
63          } catch (NumberFormatException  iae)  {
64              stdErr.println("Employee data format error!"  + iae);
65          }
66
67
68          //Testcase3
69          data = "Janne Oliver_50.00_30_500.0,1000.0,800.0,1600.0";
70
71          employee = new Employee(data);
72
73          try {
74              double expectedResultOfTestCase3 = 1578.0;
75              if (employee.getEarnings() == expectedResultOfTestCase3) {
76                  stdErr.println("Test 3 passed");
77              } else {
78                  stdErr.println("Test 3 failed");
79              }
80          } catch (NumberFormatException  iae)  {
81              stdErr.println("Employee data format error!" + iae);
82          }
83      }
```

4.4 Eclipse 重构菜单项简介



名称	作用范围	描述
Rename		可以对任意变量、类、方法、包名、文件夹进行重新命名，并且所有使用到的地方会统一进行修改。
Move	字段	把字段移到其他类、把类移到其他包
Change Method Signature	方法	对方法进行操作，可以修改方法名、访问权限、增加删除方法参数、修改参数顺序、添加方法异常
Extract Method	方法	任意选中一块代码，自动转换为方法，自动添加参数返回类型。
Extract Local Variable	字符/数字	通常用于表达式，把其中一个抽取为本地的变量，例如 3 + 5 抽取为 int i = 3;
Extract	字符/数字	把任意位置的字符串或者数字抽取为一个静态全局常量。所有

Constant		使用此字符或者数字的也会相应的被替换为使用常量。
Inline	方法	把调用此方法的地方直接替换成此方法的内容。选中任意方法才可使用此功能。（有 All invocations 与 Only the selected invocation 两个选项）
Convert Local Variable to Field	局部变量	把局部变量转变为全部变量，可以重新修改变量名。
Convert Anonymous Class to Nested	匿名类	可以设置类型，内部包含字段类型等。
Move Type to New File	嵌套类	以嵌套类创建一个新的类文件
Extract Supperclass		提取选中字段或方法放置到其父类中（注意提取方法时，先提取其中使用的字段）
Extract Interface		从一个类的方法生成一个接口（仅当前类的方法会变为接口，其他使用此方法的地方不会）
Use Supertype Where Possible		把选中引用向上转型，变成其父类的引用
Push Down		把选中方法从父类移到子类中，父类中响应方法变为抽象方法
Pull Up		与 Push Down 相反，把子类的方法上移到父类中
Extract Class	字段	把所有选中字段提到新类中，可以选择新建文件也可以使内部类
Introduce Parameter Object	方法	把方法参数抽取为一个类（避免参数在方法内引用被修改异常）
Introduce Indirection		让其他类可调用当前类某方法
Introduce Factory	构造函数	用方法返回一个对象
Introduce Parameter		将字段抽取为方法中的参数
Encapsulate Filed		为字段提供 setter/getter 方法
Generalize Declared Type		把非原始对象字段（或方法参数）替换为其父类型。
Infer Generic Type Arguments		为原始形式的那些类型推测恰当的泛型类型

Migrate JAR File		更新引用的累垮
Create Script		把当前的软件重构策略导出生成软件重构脚本
Apply Script		复用已有的软件重构策略

例 1：Convert Anonymous Class to Nested

Convert Anonymous Class to Nested 重构能够接受一个匿名类并将其转换为最初包含这个匿名类的方法的一个嵌套类。

要使用这个重构，请将光标放入这个匿名类并从菜单中选择 **Refactor > Convert Anonymous Class to Nested**。这时会出现一个对话框，要求输入新类的名称。此外，还可以设置类的属性，比如指定对这个类的访问是公共的、受保护的、私有的还是默认的。也可以指定这个类是终态的、静态的还是两者都是。

例如，清单 1 所示的代码使用一个匿名类创建了一个 Thread Factory。

清单 1. 在执行 Convert Anonymous Class to Nested 重构前

```
void createPool() {
    threadPool = Executors.newFixedThreadPool(1, new ThreadFactory()
        {

            @Override
            public Thread newThread(Runnable r)
            {
                Thread t = new Thread(r);
                t.setName("Worker thread");
                t.setPriority(Thread.MIN_PRIORITY);
                t.setDaemon(true);
                return t;
            }

        });
}
```

如果这个匿名类可被作为一个内部类单独放置，那么清单 1 中的代码将会简洁很多。因此，我执行 Convert Anonymous Class to Nested 重构，并将这个新类命名为 MyThreadFactory。结果更为简洁，如清单 2 中的代码所示。

清单 2. 执行 Convert Anonymous Class to Nested 重构后

清单 2. 执行 Convert Anonymous Class to Nested 重构后

```
private final class MyThreadFactory implements ThreadFactory
    {
        @Override
        public Thread newThread(Runnable r)
        {
            Thread t = new Thread(r);
            t.setName("Worker thread");
            t.setPriority(Thread.MIN_PRIORITY);
            t.setDaemon(true);
            return t;
        }
    }
void createPool(){
    threadPool = Executors.newFixedThreadPool(1, new MyThreadFactory());
}
```

例 2：Extract Method

重构允许您选择一块代码并将其转换为一个方法。Eclipse 会自动地推知方法参数及返回类型。如果一个方法太大并且想要把此方法再细分为不同的方法，这个重构将很有用。

如果有一段代码在很多方法中反复使用，这个重构也能派上用场。当选择这些代码块中的某一个代码块进行重构时，Eclipse 将找到出现这个代码块的其他地方，并用一个对这个新方法的调用替代它。

要使用 Extract Method 重构模式，选择编辑器中的一个代码块，按下 Alt+Shift+M。这时会出现一个对话框，要求输入这个新方法的名称及可见性（公开的、私有的、保护的或是默认的）。甚至可以更改参数和返回类型。当重构了新方法内的所选代码块以便恰当使用新方法的参数和返回类型后，新方法就创建完成了。首先完成重构的那个方法现在包括了一个对新方法的调用。例如，假设我想要在调用了清单 3 中的 map.get() 后，将代码块移到另外一个方法。

清单 3. Extract Method 重构前

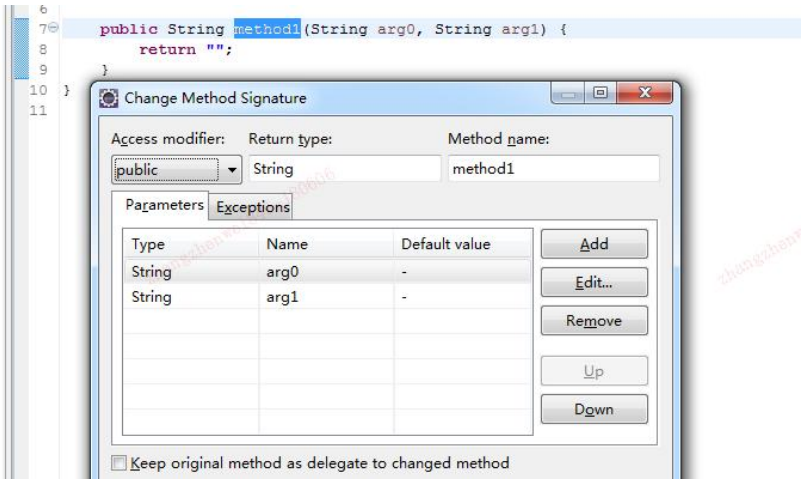
```
@Override
    public Object get(Object key)
    {
        TimedKey timedKey = new TimedKey(System.currentTimeMillis(), key);
        Object object = map.get(timedKey);

        if (object != null)
        {
            /**
             * if this was removed after the 'get' call by the worker thread
             * put it back in
             */
            map.put(timedKey, object);
            return object;
        }

        return null;
    }
```

清单 3. Extract Method 重构后

```
@Override
    public Object get(Object key)
    {
        TimedKey timedKey = new TimedKey(System.currentTimeMillis(), key);
        Object object = map.get(timedKey);

        return putIfNotNull(timedKey, object);
    }

    private Object putIfNotNull(TimedKey timedKey, Object object)
    {
        if (object != null)
        {
            /**
             * if this was removed after the 'get' call by the worker thread
             * put it back in
             */
            map.put(timedKey, object);
            return object;
        }

        return null;
    }
```

例 3：Change Method Signature（修改方法签名）

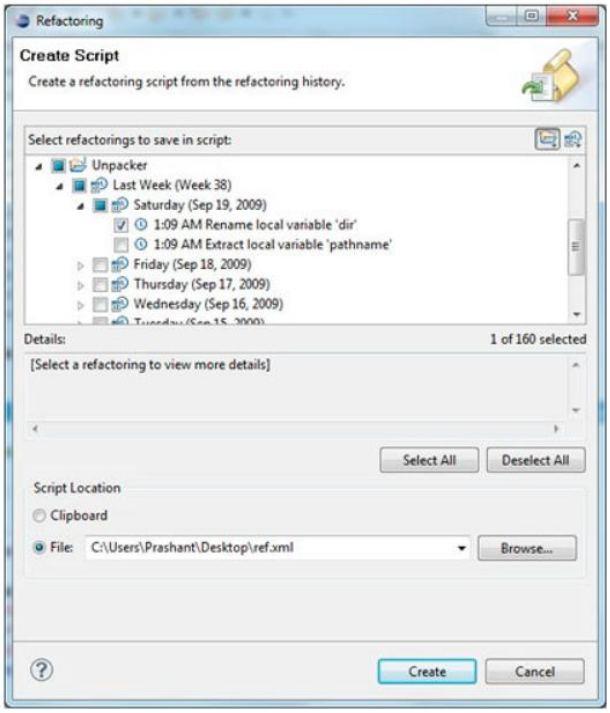
这个功能在重构时很有用，修改方法签名包括方法的作用域，返回类型，名称，入参的类型、名称、数量、顺序。而且如果是在接口或父类上改了，它的实现类和子类也会跟着改。需要注意的是，如果增加了一个参数，那么这个方法原来调用的地方默认会传 null，这很可能不是我们想要的。



例 4：重构脚本导出

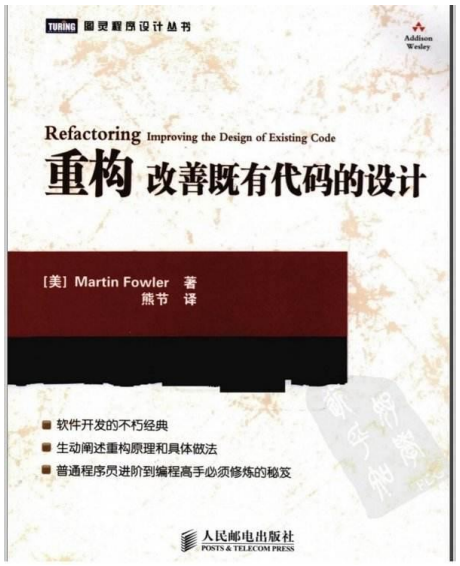
重构脚本可以让您导出并共享重构动作。当打算发布某个库的一个新版本并且人们在使用旧版本会导致错误时，重构脚本就显得很有用了。通过在发布此库的同时发布一个重构脚本，使用旧版本的人只需将这个脚本应用于其项目，就可以使其代码使用这个新版本的库了。

要创建一个重构脚本，请选择 Refactor > Create Script。这时会出现一个如下图所示的窗口，显示了在这个工作区所执行过的所有重构的历史记录。选择需要的那些重构，然后为将要生成的脚本指定一个位置，再单击 Create 生成这个脚本。



要将已有的重构脚本应用于工作区，请选择 Refactor > Apply Script。在出现的对话框中选择脚本的位置。单击 Next 以查看脚本将要执行的重构，然后单击 Finish 来应用这些重构。

软件重构技术大全，请查阅参考文献[3]:



任务 1:
以宿舍管理系统为测试对象，建立一个 baseline，用 SourceMonitor 扫码后，得到项目中 Java 源码的度量值，注意截图。

任务 2:
对宿舍管理系统，用 Ali-CodeAnalysis 进行程序静态分析，根据 CodeAnalysis 反馈结果和阿里巴巴编码规范，进行代码重构（修改代码）
代码重构后，再次用 Ali-CodeAnalysis 进行代码扫码，确保先前扫码时候发现的所有代码坏味道都被消除掉。
详细描述发现的代码坏味道？如何重构代码消除代码坏味道的？

任务 3：
对重构后的代码，建立 baseline2，记录代码重构后的代码度量指标。

任务 4：
比较代码重构前后，程序代码度量指标的变化，描述代码度量、程序静态测试和代码重构对提高代码质量的作用。

实验报告要求

编写电子稿实验报告（PDF 格式），描述实验目标、相关知识、过程、实验结果（包括阶段性结果）、完成任务 1~4、实验体会。
代码重构前后的的源码项目（包括数据库 SQL）打包为 Zip 由学委收齐，命名方式为

“Z09421XYY-姓名-软件静态测试对象.zip”。

参考文献

[1] Diomidis Spinellis. 代码阅读. 电子工业出版社. 2013.
[2] 杨冠宝（孤尽）. 阿里巴巴 Java 开发手册 . 电子工业出版社. 2018.
[3] Matin Fowler. 重构：改善既有代码的设计. 人民邮电出版社. 2012.
[4] 结城浩. 图解设计模式. 人民邮电出版社. 2016.
[5] Girish Suryanarayana, Ganesh Samarthyam, Tushar Sharma. 软件设计重构. 人民邮电出版社. 2016.
[6] 李必信，廖力，王璐璐，孔祥龙. 软件架构理论与实践. 机械工业出版社. 2019

软件下载

Visustin v8.07 Demo
https://www.aivosto.com/download.html

SourceMonitor
http://www.campwoodsw.com/sourcemonitor.html

Eclipse
建议下载 Eclipse 2018-09 (Version: 4.9.0)
http://www.eclipse.org

Ali-CodeAnalysis（Eclipse 插件）
https://p3c.alibaba.com/plugin/eclipse/update