技术为王　服务至上

软件测试与质量保证

4.2 错误定位与程序切片　张宇霞 副研究员

目录

CONTENTS

01 错误定位

02 程序切片

03 基于测试用例的错误定位

04 小结

目录

CONTENTS

01 错误定位

02 程序切片

03 基于测试用例的错误定位

04 小结

01 错误定位

■ 什么是错误定位（Fault Localization）

- ➢ 错误定位是找出程序错误的准确位置的行为。

■ 为什么要进行错误定位

- ➢ 软件测试只告诉你对错。
- ➢ 不知道错在哪里。
  - 几千上万行的代码，到底哪一行是错的?

01 错误定位

■ 错误定位的基本方法

- 断点和调试

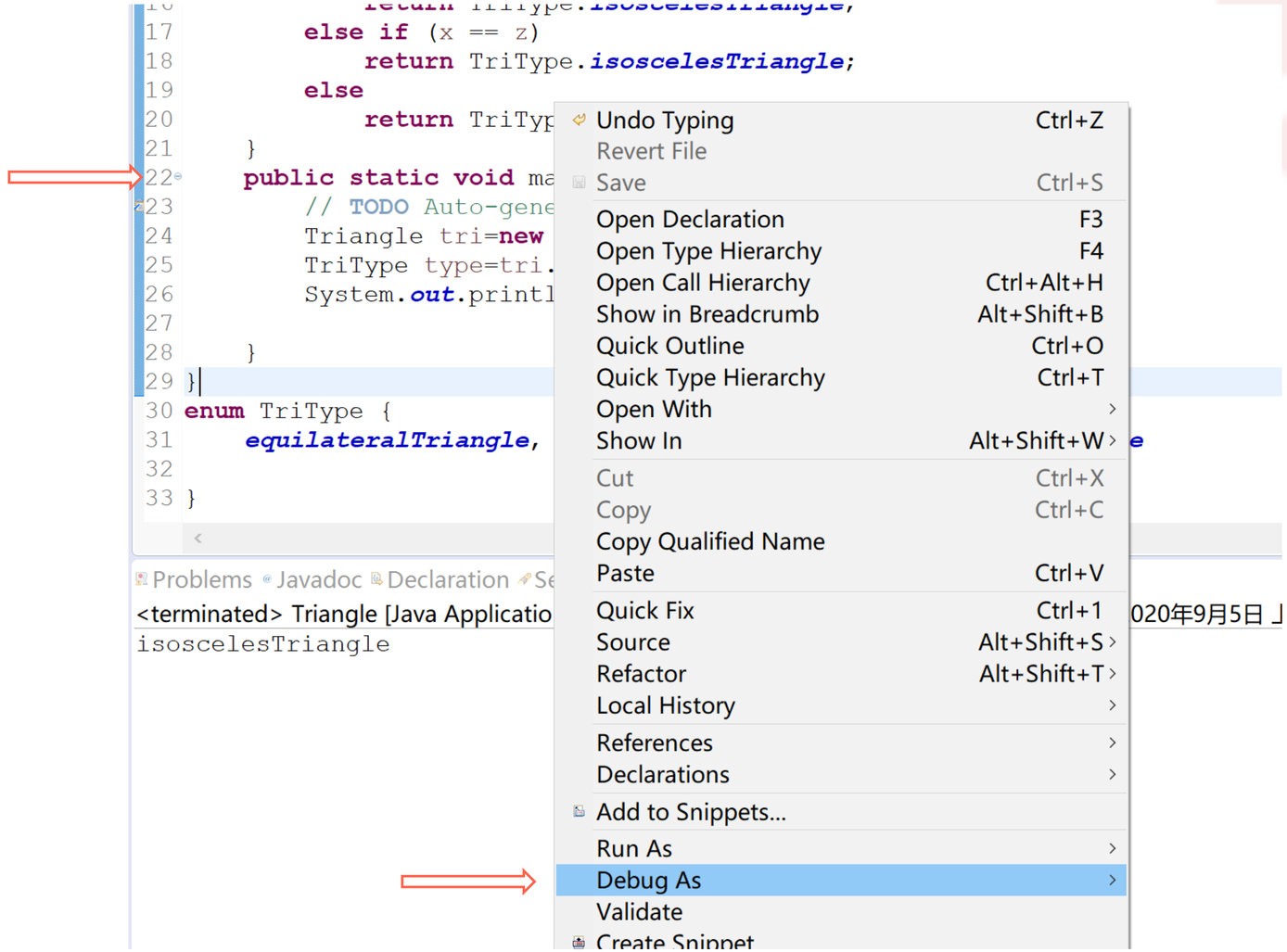
01 错误定位

```java
package dubo;
public class Triangle {
    public TriType ReturnTriangleType(int x, int y, int z) {
        if (x >= y + z)
            return TriType.noTriangle;
        if (y >= x + z)
            return TriType.noTriangle;
        if (z >= x + y)
            return TriType.noTriangle;
        if (x >= y)
            if (x == z)
                return TriType.equilateralTriangle;
            else
                return TriType.isoscelesTriangle;
        else if (y == z)
            return TriType.isoscelesTriangle;
        else if (x == z)
            return TriType.isoscelesTriangle;
        else
            return TriType.triangle;
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Triangle tri=new Triangle();
        TriType type=tri.ReturnTriangleType(12, 10, 15);
        System.out.println(type.toString());

    }
}
enum TriType {
    equilateralTriangle, isoscelesTriangle, triangle, noTriangle

}
```

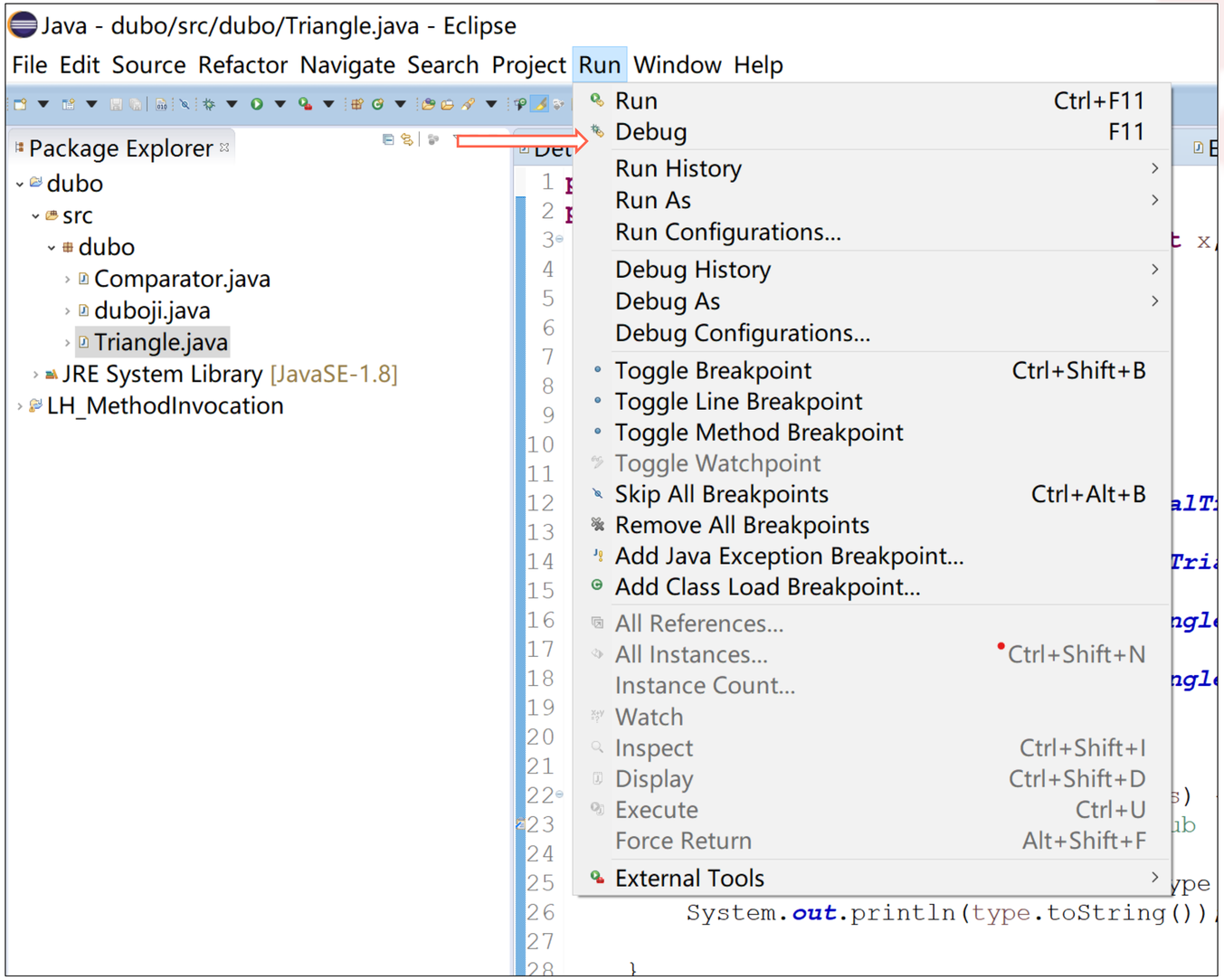
Problems Javadoc Declaration Search Console

```
<terminated> Triangle [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (2020年9月5日 」
isoscelesTriangle
```

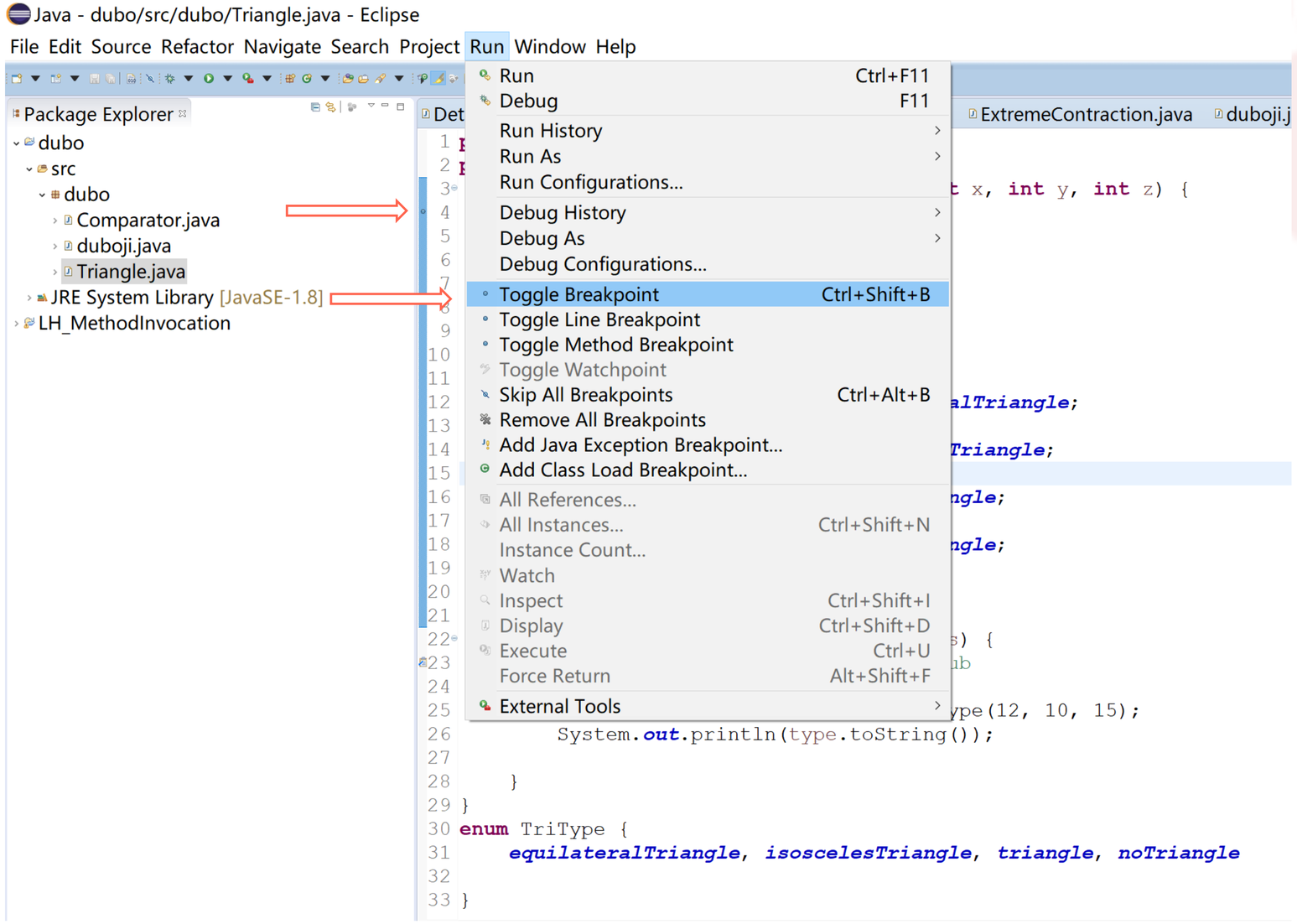
01 错误定位


17 else if (x == z)
18 return TriType.isoscelesTriangle;
19 else
20 return TriTyp
21 }
22 public static void ma
23 // TODO Auto-gene
24 Triangle tri=new
25 TriType type=tri.
26 System.out.printl
27
28 }
29 }
30 enum TriType {
31 equilateralTriangle,
32
33 }
Problems Javadoc Declaration
<terminated> Triangle [Java Applicatio
020年9月5日 」
isoscelesTriangle
Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Local History
References
Declarations
Add to Snippets...
Run As
Debug As
Validate
Create Snippet

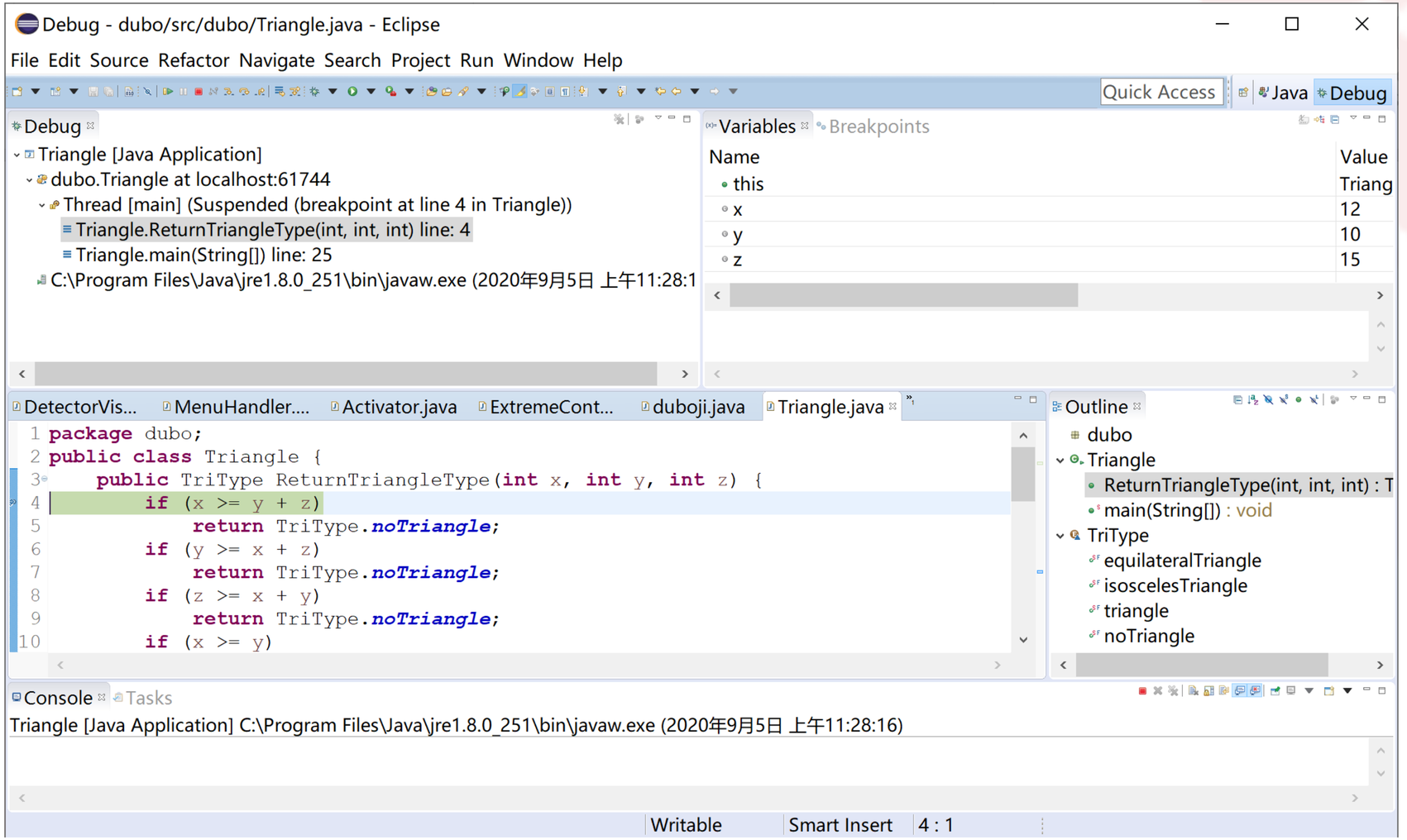
01 错误定位


Java - dubo/src/dubo/Triangle.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
dubo
src
dubo
Comparator.java
duboji.java
Triangle.java
JRE System Library [JavaSE-1.8]
LH_MethodInvocation
Run Ctrl+F11
Debug F11
Run History
Run As
Run Configurations...
Debug History
Debug As
Debug Configurations...
Toggle Breakpoint Ctrl+Shift+B
Toggle Line Breakpoint
Toggle Method Breakpoint
Toggle Watchpoint
Skip All Breakpoints Ctrl+Alt+B
Remove All Breakpoints
Add Java Exception Breakpoint...
Add Class Load Breakpoint...
All References...
All Instances... Ctrl+Shift+N
Instance Count...
Watch
Inspect Ctrl+Shift+I
Display Ctrl+Shift+D
Execute Ctrl+U
Force Return Alt+Shift+F
External Tools
System.out.println(type.toString());

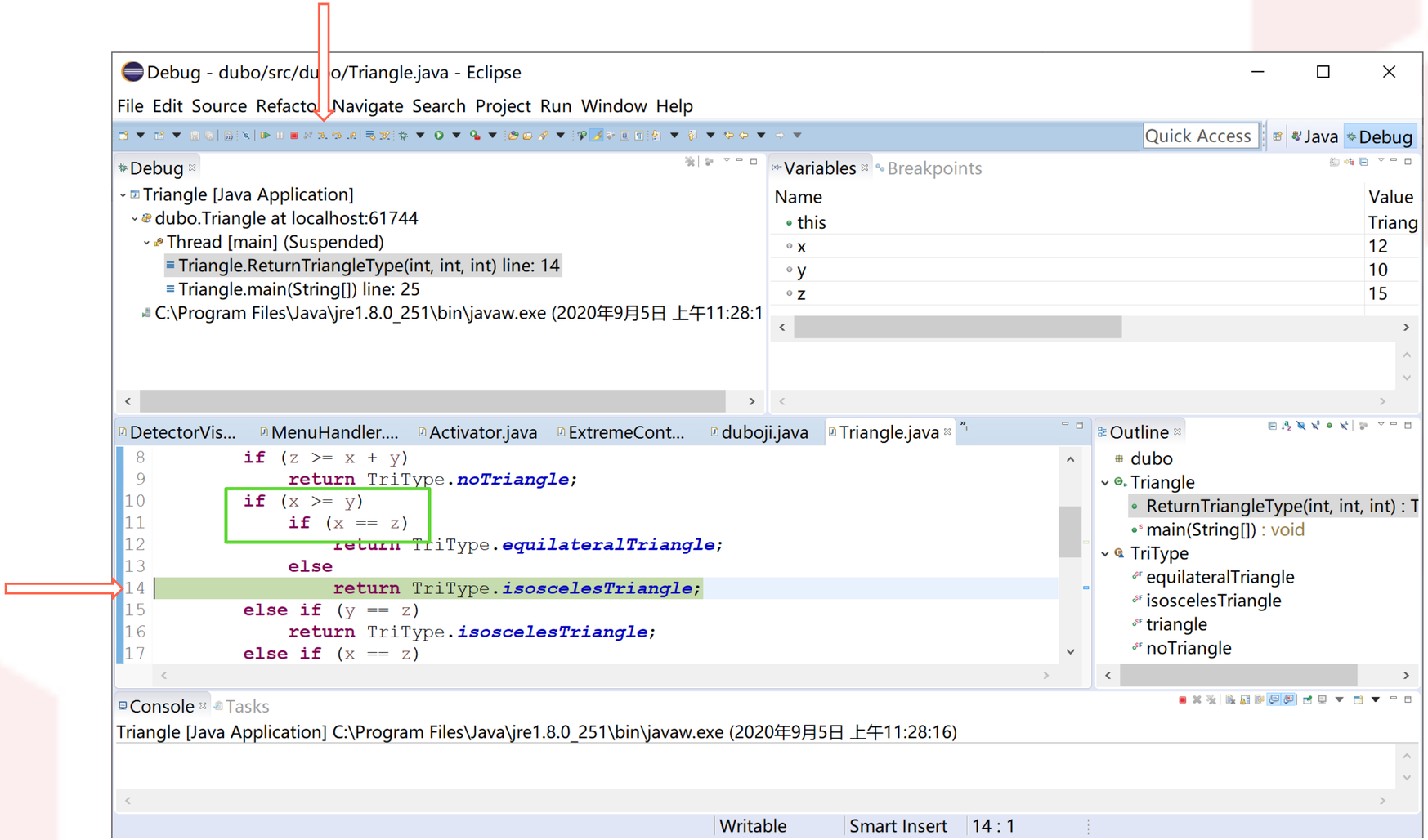
01 错误定位


Java - dubo/src/dubo/Triangle.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
dubo
src
dubo
Comparator.java
duboji.java
Triangle.java
JRE System Library [JavaSE-1.8]
LH_MethodInvocation
Run Ctrl+F11
Debug F11
Run History
Run As
Run Configurations...
Debug History
Debug As
Debug Configurations...
Toggle Breakpoint Ctrl+Shift+B
Toggle Line Breakpoint
Toggle Method Breakpoint
Toggle Watchpoint
Skip All Breakpoints Ctrl+Alt+B
Remove All Breakpoints
Add Java Exception Breakpoint...
Add Class Load Breakpoint...
All References...
All Instances... Ctrl+Shift+N
Instance Count...
Watch
Inspect Ctrl+Shift+I
Display Ctrl+Shift+D
Execute Ctrl+U
Force Return Alt+Shift+F
External Tools
ExtremeContraction.java
duboji.j
t x, int y, int z) {
alTriangle;
Triangle;
ngle;
ngle;
s) {
ub
ype(12, 10, 15);
System.out.println(type.toString());
}
}
enum TriType {
equilateralTriangle, isoscelesTriangle, triangle, noTriangle
}

01 错误定位


Debug - dubo/src/dubo/Triangle.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
Triangle [Java Application]
dubo.Triangle at localhost:61744
Thread [main] (Suspended (breakpoint at line 4 in Triangle))
Triangle.ReturnTriangleType(int, int, int) line: 4
Triangle.main(String[]) line: 25
C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (2020年9月5日 上午11:28:1
Variables
Breakpoints
Name
Value
this
Triang
x
12
y
10
z
15
DetectorVis...
MenuHandler....
Activator.java
ExtremeCont...
duboji.java
Triangle.java
1 package dubo;
2 public class Triangle {
3 public TriType ReturnTriangleType(int x, int y, int z) {
4 if (x >= y + z)
5 return TriType.noTriangle;
6 if (y >= x + z)
7 return TriType.noTriangle;
8 if (z >= x + y)
9 return TriType.noTriangle;
10 if (x >= y)
Outline
dubo
Triangle
ReturnTriangleType(int, int, int) : T
main(String[]) : void
TriType
equilateralTriangle
isoscelesTriangle
triangle
noTriangle
Console
Tasks
Triangle [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (2020年9月5日 上午11:28:16)
Writable
Smart Insert
4 : 1

01 错误定位


Debug - dubo/src/dubo/Triangle.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
Triangle [Java Application]
dubo.Triangle at localhost:61744
Thread [main] (Suspended)
Triangle.ReturnTriangleType(int, int, int) line: 14
Triangle.main(String[]) line: 25
C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (2020年9月5日 上午11:28:1
Variables
Breakpoints
Name
Value
this
Triang
x
12
y
10
z
15
DetectorVis...
MenuHandler....
Activator.java
ExtremeCont...
duboji.java
Triangle.java
8 if (z >= x + y)
9 return TriType.noTriangle;
10 if (x >= y)
11 if (x == z)
12 return TriType.equilateralTriangle;
13 else
14 return TriType.isoscelesTriangle;
15 else if (y == z)
16 return TriType.isoscelesTriangle;
17 else if (x == z)
Outline
dubo
Triangle
ReturnTriangleType(int, int, int) : T
main(String[]) : void
TriType
equilateralTriangle
isoscelesTriangle
triangle
noTriangle
Console
Tasks
Triangle [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (2020年9月5日 上午11:28:16)
Writable
Smart Insert
14 : 1

01 错误定位

```
64  private void detectProject() {
65      IProject[] projects = ResourcesPlugin.getWorkspace().getRoot().getProjects();
66      for(IProject project : projects){
67          IJavaProject javaProject = JavaCore.create(project);
68          System.out.println("List of Method Invocations on Project: \t" + javaProject.getElement
69          try {
70              IPackageFragment[] fragments = javaProject.getPackageFragments();
71               for(IPackageFragment fragment : fragments){
72                  ICompilationUnit[] compilationUnits = fragment.getCompilationUnits();
73                  for(ICompilationUnit compilationUnit : compilationUnits){
74                      System.out.println(compilationUnit);
75                      ASTParser astParser = ASTParser.newParser(AST.JLS4);
76                      astParser.setKind(ASTParser.K_COMPILATION_UNIT);
77                      astParser.setResolveBindings(true);
78                      astParser.setBindingsRecovery(true);
79                      astParser.setSource(compilationUnit);
80                      CompilationUnit unit = (CompilationUnit) (astParser.createAST(null));
81          /*如果无法获得调用method，可以这样获取
82           *      List<org.eclipse.jdt.core.dom.TypeDeclaration> types=unit.types();
```

01 错误定位

■ 插桩（Instrumentation）

- 插桩是在软件特定的位置插入一些语句，用来获取软件的运行信息并反馈给测试者。
- 又称软件探针(Software Probe)。

01 错误定位

■ 错误定位的基本方法

- 断点和调试
- 猜测

01 错误定位

■ 最可能出错的语句

- 函数调用语句。子函数的调用语句是测试的重点，一方面由于在调用子函数时可能引起接口引用错误，另一方面可能是子函数本身的错误。
- 判定转移/循环语句。判定语句常常会由于边界值与比较优先级等问题引起错误或失效而做出错误的转移。因此，对于判定转移/循环语句也是一个重要的测试点。

01 错误定位

■ 最可能出错的语句

- 复杂算法段。出错的概率常与算法的复杂度成正比。所以越复杂的算法越需要作重点跟踪，如递归、回朔等算法。
- SQL语句。对于数据库的应用程序来说，SQL语句常常会在模块中占比较重要的业务逻辑，而且比较复杂。因此，它也属于比较容易出现错误的语句。

01 错误定位

■ SQL语句执行检查

- ➢ 捕获并检查生成的SQL语句是否正确。
- ➢ 将SQL脚本放入数据库执行看效果。

01 错误定位

- **错误定位的基本方法**
  - 断点和调试
  - 猜测
- **错误定位的科学方法**
  - 程序切片

目录

CONTENTS

01 错误定位

02 **程序切片**

03 基于测试用例的错误定位

04 小结

02 程序切片

■ 什么是程序切片

- 与某个输出有关的所有语句和谓词构成的程序称为源程序的一个切片。

02

程序切片

■ 程序切片的准确计算

➢ 给定一个程序P和P中的一个变量集合V，变量集合V在语句n上的一个切片，记做S（V，n），是P中对V的变量值作出贡献的所有语句集合。

- 赋值定义变量v
- 对v变量的赋值所使用的变量具有影响的语句
- 与变量v相关的控制流控制语句

02

程序切片

```
1  Program Commission(INPUT,OUTPUT)
2  Dim locks,stocks,barrels as Integer
3  Dim lockPrice,stockPrice,barrelPrice as Real
4  Dim totalLocks,totalStocks,totalBarrels as Integer
5  Dim lockSales,stockSales,barrelSales as Real
6  Dim sales,commission as Real
7  lockPrice=45.0
8  stockPrice=30.0
9  barrelPrice=25.0
10 totalLocks=0
11 totalStocks=0
12 totalBarrels=0
13 Input(locks)
14 While(locks)
15       Input(stocks,barrels)
16       totalLocks=totalLocks+locks
17       totalStocks=totalStocks+stocks
18       totalBarrels=totalBarrels+barrels
19       Output(“Locks:“,locks)
20       Input(locks)
20 EndWhile
21 Output(“Locks sold:“,totalLocks)
```

02

程序切片

变量locks切片：

S1:S(locks,13)={13}

S2:S(locks,14)={13,14,20,21}

S3:S(locks,16)={13,14,20,21}

S4:S(locks,19)={13,14,20,21}

S5:S(locks,20)={20}

```
1  Program Commission(INPUT,OUTPUT)
2  Dim locks,stocks,barrels as Integer
3  Dim lockPrice,stockPrice,barrelPrice as Real
4  Dim totalLocks,totalStocks,totalBarrels as Integer
5  Dim lockSales,stockSales,barrelSales as Real
6  Dim sales,commission as Real
7  lockPrice=45.0
8  stockPrice=30.0
9  barrelPrice=25.0
10 totalLocks=0
11 totalStocks=0
12 totalBarrels=0
13 Input(locks)
14 While(locks)
15       Input(stocks,barrels)
16       totalLocks=totalLocks+locks
17       totalStocks=totalStocks+stocks
18       totalBarrels=totalBarrels+barrels
19       Output("Locks:",locks)
20       Input(locks)
21 EndWhile
22 Output("Locks sold:",totalLocks)
```

02 程序切片

■ 程序切片有什么用?

- 与某个输出有关的所有语句和位置构成的程序称为源程序的一个切片。
- 如果某个变量与预期不一致，那么一定是它的切片中的某个（些）语句有问题。
- 切片技术可以大幅缩写错误定位的范围。

02

程序切片

■ **程序切片有什么用?**

- z的实际值与预期值不一致。
- 可能的错误语句有哪些?

① x = 1;

② y = 2;

③ z = y-2;

④ r = x;

⑤ z = x+y;

02

程序切片

■ **程序切片有什么用?**

- z的实际值与预期值不一致。
- 可能的错误语句有哪些?
  - **1、2、5**

① x = 1;

② y = 2;

③ z = y-2;

④ r = x;

⑤ z = x+y;

目录

CONTENTS

01 错误定位

02 程序切片

03 **基于测试用例的错误定位**

04 小结

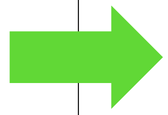
03 基于测试用例的错误定位

■ 测试用例

- 某个给定的程序输入X、预期输出Y、真实输出Y′ 。
- 通过测试用例，获知某个程序（某个函数）是否有错。
- 有错的程序/函数，并不意味着它的每个测试都会出错！

03 基于测试用例的错误定位

```
int mid(int x, int y, int z) {
    int m;
    m= z;
    if (y < z) {
      if (x < y) m = y;
      else if (x < z) m = y;
} else {
        if (x > y) m = y;
        else if ( x > z) m = x;
      }
    } return m;
}
```



m=x

03 基于测试用例的错误定位

语句	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
`int m;`	√	√	√	√	√	√
`m = z;`	√	√	√	√	√	√
`if (y < z) {`	√	√	√	√	√	√
`if (x < y)`		√			√	√
`m = y;`		√				
`else if (x < z)`	√				√	√
`m = y;`	√					√
`} else {`			√	√		
`if (x > y)`			√	√		
`m = y;`			√			
`else if (x > z)`				√		
`m = x; }`						
`return m;`	√	√	√	√	√	√
	成功	成功	成功	成功	成功	失败

03 基于测试用例的错误定位

■ 每个语句的可疑度

- ➢ 经过该语句的失败测试用例数目越多，它就越可疑。
- ➢ 经过该语句的成功测试用例数目越多，它就越不可疑。

■ 可疑度排序

- ➢ 经过两条语句的失败测试用例数目一样的情况下，随经过的成功测试用例数目更少，它就更可疑。
- ➢ 经过两条语句的成功测试用例数目一样的情况下，随经过的失败测试用例数目更多，它就更可疑。

03 基于测试用例的错误定位

语句	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
`int m;`	√	√	√	√	√	√
`m = z;`	√	√	√	√	√	√
`if (y < z) {`	√	√	√	√	√	√
`if (x < y)`		√			√	√
`m = y;`		√				
`else if (x < z)`	√				√	√
`m = y;`	√					√
`} else {`			√	√		
`if (x > y)`			√	√		
`m = y;`			√			
`else if (x > z)`				√		
`m = x; }`						
`return m;`	√	√	√	√	√	√
	成功	成功	成功	成功	成功	失败

03 基于测试用例的错误定位

■ 可疑度计算

➢ 语句s的可疑度Sus(s)

$$Sus(s) = \frac{fail(s) / totalfail}{fail(s) / totalfail + pass(s) / totalpass}$$

03 基于测试用例的错误定位

语句	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	Sus(s)
`int m;`	√	√	√	√	√	√	
`m = z;`	√	√	√	√	√	√	
`if (y < z) {`	√	√	√	√	√	√	
`if (x < y)`		√			√	√	
`m = y;`		√					
`else if (x < z)`	√				√	√	
`m = y;`	√					√	
`} else {`			√	√			
`if (x > y)`			√	√			
`m = y;`			√				
`else if (x > z)`				√			
`m = x; }`							
`return m;`	√	√	√	√	√	√	
	成功	成功	成功	成功	成功	失败	

目录

CONTENTS

01 错误定位

02 程序切片

03 基于测试用例的错误定位

04 小结

04 小结

错误定位是修正软件缺陷的必经之路

错误定位需要高度的技巧

程序切片和测试用例都是错误定位的有力武器

谢谢