

# Introduction to Code-Motion Refactoring

## Final Project

```
if (settings[0] != null) {
    if (name.compareTo(settings[0]) != 0) {
        name += " - ";
    }
    name += etr.getString(settings[0]);
    if (settings[0].compareTo("") != 0) {
        name += " - ";
    }
    name += DateUtils.format(etr.getDate(settings[0]));
    if (settings[0].compareTo("") != 0) {
        name += " - ";
    }
    name += etr.getDouble(settings[0]);
    if (settings[0].compareTo("") != 0) {
        name += " - ";
    }
}

print "ncfiles: vlling etr"
if (name.compareTo("") != 0) {
    name += " - ";
}
name += DateUtils.format(etr.getDate(settings[0]));
if (settings[0].compareTo("") != 0) {
    name += " - ";
}
name += etr.getDouble(settings[0]);
if (settings[0].compareTo("") != 0) {
    name += " - ";
}

age.findAll("h3");
(h3.contents[0])
if (h3.contents[0] != "Afdeling"):
    print "ncfiles: Socket error"
    print "ncfiles: Socket error"

port codecs
= codecs.open("alle.txt", "r", encoding="utf-8")
text = f.read()
f.close()
# open the file again for writing
f = codecs.open("alle.txt", "w", encoding="utf-8")
f.write(value+"\n")
# write the original contents
f.write(text)
```

### Sliding:

We chose a method called "findRangBounds" from an open source project which can be found at:

<https://github.com/alwaqfi/jfreechart-1.0.10/blob/master/src/org/jfree/chart/renderer/xy/VectorRenderer.java>

[jfreechart-1.0.10/src/org/jfree/chart/renderer/xy/VectorRenderer.java](https://github.com/alwaqfi/jfreechart-1.0.10/blob/master/src/org/jfree/chart/renderer/xy/VectorRenderer.java)

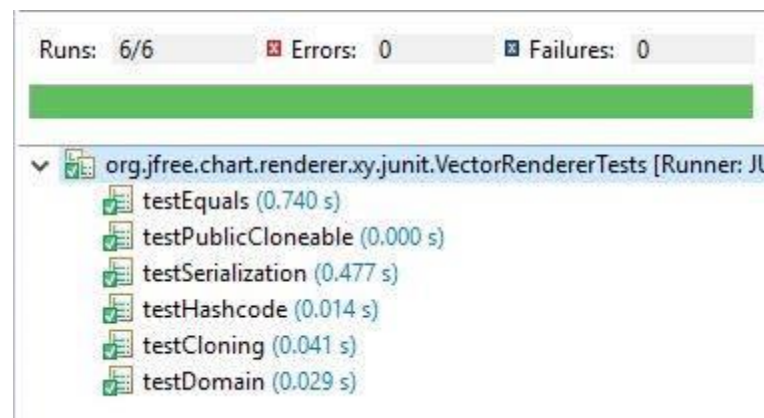
### original method:

```
95 public Range findDomainBounds(XYDataset dataset) {
96     if (dataset == null) {
97         throw new IllegalArgumentException("Null 'dataset' argument.");
98     }
99     double minimum = Double.POSITIVE_INFINITY;
100     double maximum = Double.NEGATIVE_INFINITY;
101     int seriesCount = dataset.getSeriesCount();
102     double lvalue;
103     double uvalue;
104     if (dataset instanceof VectorXYDataset) {
105         VectorXYDataset vdataset = (VectorXYDataset) dataset;
106         for (int series = 0; series < seriesCount; series++) {
107             int itemCount = dataset.getItemCount(series);
108             for (int item = 0; item < itemCount; item++) {
109                 double delta = vdataset.getVectorXValue(series, item);
110                 if (delta < 0.0) {
111                     uvalue = vdataset.getXValue(series, item);
112                     lvalue = uvalue + delta;
113                 }
114                 else {
115                     lvalue = vdataset.getXValue(series, item);
116                     uvalue = lvalue + delta;
117                 }
118                 minimum = Math.min(minimum, lvalue);
119                 maximum = Math.max(maximum, uvalue);
120             }
121         }
122     }
123     else {
124         for (int series = 0; series < seriesCount; series++) {
125             int itemCount = dataset.getItemCount(series);
126             for (int item = 0; item < itemCount; item++) {
127                 lvalue = dataset.getXValue(series, item);
128                 uvalue = lvalue;
129                 minimum = Math.min(minimum, lvalue);
130                 maximum = Math.max(maximum, uvalue);
131             }
132         }
133     }
134     if (minimum > maximum) {
135         return null;
136     }
137     else {
138         return new Range(minimum, maximum);
139     }
140 }
```

Since this method doesn't have a test we created another method for testing, please add the method in here:

[jfreechart-1.0.10/tests/org/jfree/chart/renderer/xy/junit/VectorRendererTests.java](https://jfreechart-1.0.10/tests/org/jfree/chart/renderer/xy/junit/VectorRendererTests.java)

```
162 public void testDomain() {
163     DefaultXYDataset dl = new DefaultXYDataset();
164     double[] x1 = new double[] {1.0, 2.0, 3.0};
165     double[] y1 = new double[] {4.0, 5.0, 6.0};
166     double[][] data1 = new double[][] {x1, y1};
167     dl.addSeries("S1", data1);
168
169     VectorRenderer r1 = new VectorRenderer();
170     Range r2 = new Range(1.0, 3.0);
171     assertTrue(r1.findDomainBounds(dl).equals(r2));
172 }
173
174 }
```



### Step1:

we focused on a specific block of code in the method, and applied the Sliding algorithm on that specific block of code.

This block of code compute tow variables independently {maximum, minimum}.

We would like to split those computations into tow sperate methods.

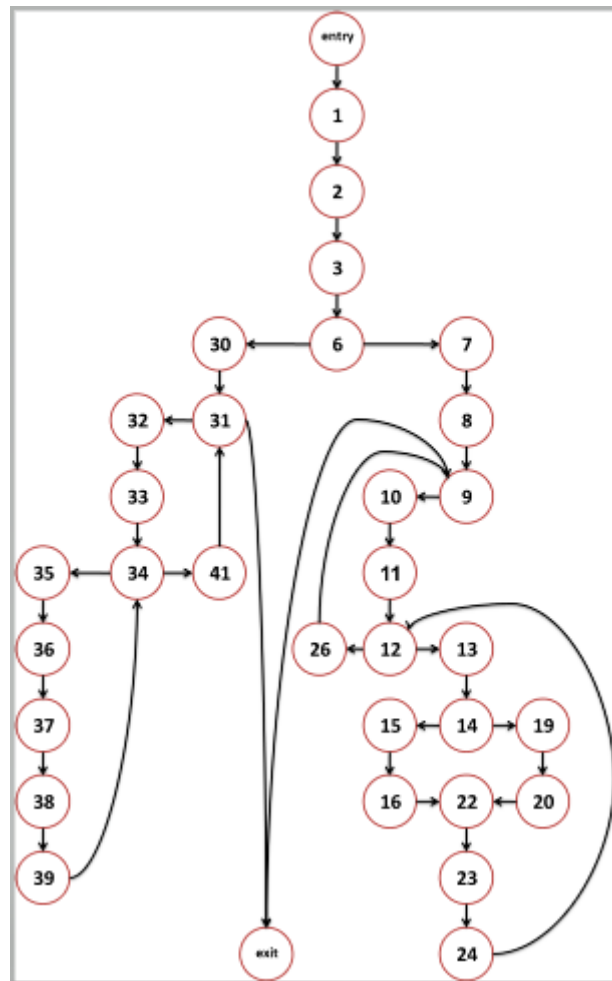
We chose the block of code from lines 99-133, for convenience we numbered those lines from 1-43 after converting "for" loops to "while" loops.

```

1  double minimum = Double.POSITIVE_INFINITY;
2  double maximum = Double.NEGATIVE_INFINITY;
3  int seriesCount = dataset.getSeriesCount();
4  double lvalue;
5  double uvalue;
6  if (dataset instanceof VectorXYDataset) {
7      VectorXYDataset vdataset = (VectorXYDataset) dataset;
8      int series = 0;
9      while (series < seriesCount){
10         int itemCount = dataset.getItemCount(series);
11         int item=0;
12         while (item < itemCount){
13             double delta = vdataset.getVectorXValue(series, item);
14             if (delta < 0.0) {
15                 uvalue = vdataset.getXValue(series, item);
16                 lvalue = uvalue + delta;
17             }
18             else {
19                 lvalue = vdataset.getXValue(series, item);
20                 uvalue = lvalue + delta;
21             }
22             minimum = Math.min(minimum, lvalue);
23             maximum = Math.max(maximum, uvalue);
24             item++;
25         }
26         series++;
27     }
28 }
29 else {
30     int series = 0;
31     while (series < seriesCount) {
32         int itemCount = dataset.getItemCount(series);
33         int item = 0;
34         while (item < itemCount){
35             lvalue = dataset.getXValue(series, item);
36             uvalue = lvalue;
37             minimum = Math.min(minimum, lvalue);
38             maximum = Math.max(maximum, uvalue);
39             item++;
40         }
41         series++;
42     }
43 }

```

### Build the CFG:



### Build the PDG:

Edge	type
(Entry,1)	<b>Control</b>
(Entry,2)	<b>Control</b>
(Entry,3)	<b>Control</b>
(Entry,6)	<b>Control</b>
(6,7)	<b>Control</b>
(6,8)	<b>Control</b>
(6,9)	<b>Control</b>
(6,30)	<b>Control</b>
(6,31)	<b>Control</b>
(9,10)	<b>Control</b>
(9,11)	<b>Control</b>
(9,12)	<b>Control</b>
(9,26)	<b>Control</b>
(12,13)	<b>Control</b>
(12,14)	<b>Control</b>
(12,22)	<b>Control</b>
(12,23)	<b>Control</b>
(12,24)	<b>Control</b>
(14,15)	<b>Control</b>
(14,16)	<b>Control</b>
(14,19)	<b>Control</b>
(14,20)	<b>Control</b>
(31,32)	<b>Control</b>
(31,33)	<b>Control</b>
(31,34)	<b>Control</b>
(31,41)	<b>Control</b>
(34,35)	<b>Control</b>
(34,36)	<b>Control</b>
(34,37)	<b>Control</b>
(34,38)	<b>Control</b>
(34,39)	<b>Control</b>
(9, exit)	<b>Control</b>
(31, exit)	<b>Control</b>

Edge	type	Vars
(1,22)	Flow	<b>{minimum}</b>
(1,37)	Flow	<b>{minimum}</b>
(2,23)	Flow	<b>{maximum}</b>
(2,38)	Flow	<b>{maximum}</b>
(3,31)	Flow	<b>{seriesCount}</b>
(3,9)	Flow	<b>{seriesCount}</b>
(8,9)	Flow	<b>{series}</b>
(8,10)	Flow	<b>{series}</b>
(8,13)	Flow	<b>{series}</b>
(8,15)	Flow	<b>{series}</b>
(8,19)	Flow	<b>{series}</b>
(8,26)	Flow	<b>{series}</b>
(26,9)	Flow	<b>{series}</b>
(26,10)	Flow	<b>{series}</b>
(26,13)	Flow	<b>{series}</b>
(26,15)	Flow	<b>{series}</b>
(26,19)	Flow	<b>{series}</b>
(26,26)	Flow	<b>{series}</b>
(30,31)	Flow	<b>{series}</b>
(30,32)	Flow	<b>{series}</b>
(30,35)	Flow	<b>{series}</b>
(30,41)	Flow	<b>{series}</b>
(41,32)	Flow	<b>{series}</b>
(41,35)	Flow	<b>{series}</b>
(41,41)	Flow	<b>{series}</b>
(41,31)	Flow	<b>{series}</b>
(16,22)	Flow	<b>{lvalue}</b>
(19,22)	Flow	<b>{lvalue}</b>
(19,20)	Flow	<b>{lvalue}</b>
(35,36)	Flow	<b>{lvalue}</b>
(35,37)	Flow	<b>{lvalue}</b>
(34,38)	Flow	<b>{uvalue}</b>
(15,16)	Flow	<b>{uvalue}</b>

Edge	type	Vars
(1, exit)	Flow	{minimum}
(22, exit)	Flow	{minimum}
(37, exit)	Flow	{minimum}
(2, exit)	Flow	{maximum}
(23, exit)	Flow	{maximum}
(38, exit)	Flow	{maximum}

Edge	type	Vars
(15,23)	Flow	{uvalue}
(20,23)	Flow	{uvalue}
(36,38)	Flow	{uvalue}
(10,12)	Flow	{itemCount}
(32,34)	Flow	{itemCount}
(32,34)	Flow	{item}
(11,12)	Flow	{item}
(11,13)	Flow	{item}
(11,15)	Flow	{item}
(11,19)	Flow	{item}
(11,24)	Flow	{item}
(24,12)	Flow	{item}
(24,13)	Flow	{item}
(24,15)	Flow	{item}
(24,19)	Flow	{item}
(24,24)	Flow	{item}
(33,34)	Flow	{item}
(33,35)	Flow	{item}
(33,39)	Flow	{item}
(39,34)	Flow	{item}
(39,35)	Flow	{item}
(39,39)	Flow	{item}
(13,14)	Flow	{delta}
(13,16)	Flow	{delta}
(13,20)	Flow	{delta}
(entry,3)	Flow	{dataset}
(entry,6)	Flow	{dataset}
(entry,7)	Flow	{dataset}
(entry,10)	Flow	{dataset}
(entry,32)	Flow	{dataset}
(entry,35)	Flow	{dataset}
(7,13)	Flow	{ vdataset }
(7,15)	Flow	{ vdataset }

Edge	type	Vars
(16,13)	anti	<b>{delta}</b>
(20,13)	anti	<b>{delta}</b>
(20,16)	anti	<b>{lvalue}</b>
(20,19)	anti	<b>{lvalue}</b>
(22,16)	anti	<b>{lvalue}</b>
(22,19)	anti	<b>{ lvalue }</b>
(36,35)	anti	<b>{ lvalue }</b>
(37,35)	anti	<b>{ lvalue }</b>
(16,20)	anti	<b>{uvalue}</b>
(16,15)	anti	<b>{uvalue}</b>
(23,15)	anti	<b>{uvalue}</b>
(23,20)	anti	<b>{uvalue}</b>
(38,36)	anti	<b>{uvalue}</b>

Edge	type	Vars
(23,23)	anti	<b>{maximum}</b>
(38,38)	anti	<b>{ maximum }</b>
(22,22)	anti	<b>{minimum}</b>
(37,37)	anti	<b>{ minimum }</b>
(9,26)	anti	<b>{ series }</b>
(10,26)	anti	<b>{ series }</b>
(13,26)	anti	<b>{ series }</b>
(15,26)	anti	<b>{ series }</b>
(19,26)	anti	<b>{ series }</b>
(26,26)	anti	<b>{ series }</b>
(31,41)	anti	<b>{ series }</b>
(32,41)	anti	<b>{series }</b>
(35,41)	anti	<b>{series }</b>
(41,41)	anti	<b>{series}</b>
(12,10)	anti	<b>{itemCount}</b>
(34,32)	anti	<b>{itemCount}</b>
(12,24)	anti	<b>{item}</b>
(15,24)	anti	<b>{item}</b>
(13,24)	anti	<b>{item}</b>
(19,24)	anti	<b>{item}</b>
(24,24)	anti	<b>{item}</b>
(34,39)	anti	<b>{item}</b>
(35,39)	anti	<b>{item}</b>
(39,39)	anti	<b>{item}</b>
(14,13)	anti	<b>{delta}</b>



## Step 2:

We apply sliding algorithm on  $V=\{\text{minimum}\}$ .

Slice:

To do the slice we remove flow dependencies to exit of any variable other than "minimum".

Now we do back tracking starting from exit node up, using control and flow dependencies.

$\text{Slice}(\text{exit}) = \{\text{entry}, 1, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 22, 24, 26, 30, 31, 32, 33, 34, 35, 37, 39, 41, \text{exit}\}$

Co- Slice:

To do the Co-Slice we remove flow dependencies created by the variable "minimum" (y, x) such that there is no anti dependence (x, z) created by the variable "minimum".

Those are the edges that we remove:

(1, exit)	Flow	{minimum}
(22, exit)	Flow	{minimum}
(37, exit)	Flow	{minimum}

$\text{Co-Slice}(\text{exit}) = \{\text{entry}, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 20, 23, 24, 26, 30, 31, 32, 33, 34, 35, 36, 38, 39, 41, \text{exit}\}$

## Compensations checking:

**Pen1:** The value of the extracted variable could change in the Co-Slice.

There is no definition to variable "minimum" in any Co-Slice's nodes, therefore **pen1 = {}**.

**Pen2:** there exists a usage in the Co-Slice of a non-final value of the extracted variable.

There is no usage to variable "minimum" in any Co-Slice's nodes, therefore **pen2 = {}**.

**Pen3:** the slice could change the value of unextracted variable, that its initial value is required at the co-slice.

There isn't Slice's nodes defining variables causing flow dependencies from entry to Co-Slice's nodes, therefore **pen3 = {}**.

The result will be:

```
95 public Range findDomainBounds(XYDataset dataset) {
96     if (dataset == null) {
97         throw new IllegalArgumentException("Null 'dataset' argument.");
98     }
99     //*****slice*****
100     double minimum = Double.POSITIVE_INFINITY;
101     int seriesCount = dataset.getSeriesCount();
102     double lvalue;
103     double uvalue;
104     if (dataset instanceof VectorXYDataset) {
105         VectorXYDataset vdataset = (VectorXYDataset) dataset;
106         int series = 0;
107         while (series < seriesCount){
108             int itemCount = dataset.getItemCount(series);
109             int item=0;
110             while (item < itemCount){
111                 double delta = vdataset.getVectorXValue(series, item);
112                 if (delta < 0.0) {
113                     uvalue = vdataset.getXValue(series, item);
114                     lvalue = uvalue + delta;
115                 }
116                 else {
117                     lvalue = vdataset.getXValue(series, item);
118                 }
119                 minimum = Math.min(minimum, lvalue);
120                 item++;
121             }
122             series++;
123         }
124     }
125     else {
126         int series = 0;
127         while (series < seriesCount) {
128             int itemCount = dataset.getItemCount(series);
129             int item = 0;
130             while (item < itemCount){
131                 lvalue = dataset.getXValue(series, item);
132                 minimum = Math.min(minimum, lvalue);
133                 item++;
134             }
135             series++;
136         }
137     }
138
139     //*****co-slice*****
140     double maximum = Double.NEGATIVE_INFINITY;
141     seriesCount = dataset.getSeriesCount();
142     if (dataset instanceof VectorXYDataset) {
143         VectorXYDataset vdataset = (VectorXYDataset) dataset;
144         int series = 0;
145         while (series < seriesCount){
146             int itemCount = dataset.getItemCount(series);
147             int item=0;
148             while (item < itemCount){
149                 double delta = vdataset.getVectorXValue(series, item);
150                 if (delta < 0.0) {
151                     uvalue = vdataset.getXValue(series, item);
152                 }
153                 else {
154                     lvalue = vdataset.getXValue(series, item);
155                     uvalue = lvalue + delta;
156                 }
157                 maximum = Math.max(maximum, uvalue);
158                 item++;
159             }
160             series++;
161         }
162     }
163     else {
164         int series = 0;
165         while (series < seriesCount) {
166             int itemCount = dataset.getItemCount(series);
167             int item = 0;
168             while (item < itemCount){
169                 uvalue = dataset.getXValue(series, item);
170                 maximum = Math.max(maximum, uvalue);
171                 item++;
172             }
173             series++;
174         }
175     }
176     return new Range(minimum, maximum);
177 }
```

```

161     }
162 }
163 else {
164     int series = 0;
165     while (series < seriesCount) {
166         int itemCount = dataset.getItemCount(series);
167         int item = 0;
168         while (item < itemCount){
169             lvalue = dataset.getXValue(series, item);
170             uvalue = lvalue;
171             maximum = Math.max(maximum, uvalue);
172             item++;
173         }
174         series++;
175     }
176 }
177
178 //end of changes
179 if (minimum > maximum) {
180     return null;
181 }
182 else {
183     return new Range(minimum, maximum);
184 }
185 }
186 }

```

### Step 3:

Extract the Slice into a new method.

```
195 public Range findDomainBounds(XYDataset dataset) {
196     if (dataset == null) {
197         throw new IllegalArgumentException("Null 'dataset' argument.");
198     }
199     //*****slice*****
200     int seriesCount;
201     double lvalue;
202     double uvalue;
203     double minimum = findMinimum(dataset);
204
205     //*****co-slice*****
206     double maximum = Double.NEGATIVE_INFINITY;
207     seriesCount = dataset.getSeriesCount();
208     if (dataset instanceof VectorXYDataset) {
209         VectorXYDataset vdataset = (VectorXYDataset) dataset;
210         int series = 0;
211         while (series < seriesCount){
212             int itemCount = dataset.getItemCount(series);
213             int item=0;
214             while (item < itemCount){
215                 double delta = vdataset.getVectorXValue(series, item);
216                 if (delta < 0.0) {
217                     uvalue = vdataset.getXValue(series, item);
218                 }
219                 else {
220                     lvalue = vdataset.getXValue(series, item);
221                     uvalue = lvalue + delta;
222                 }
223                 maximum = Math.max(maximum, uvalue);
224                 item++;
225             }
226             series++;
227         }
228     }
229     else {
230         int series = 0;
231         while (series < seriesCount) {
232             int itemCount = dataset.getItemCount(series);
233             int item = 0;
234             while (item < itemCount){
235                 lvalue = dataset.getXValue(series, item);
236                 uvalue = lvalue;
237                 maximum = Math.max(maximum, uvalue);
238                 item++;
239             }
240             series++;
241         }
242     }
243
244     //end of changes
245     if (minimum > maximum) {
246         return null;
247     }
248     else {
249         return new Range(minimum, maximum);
250     }
251 }
252
253
```

```

154 private double findMinimum(XYDataset dataset) {
155     double minimum = Double.POSITIVE_INFINITY;
156     int seriesCount = dataset.getSeriesCount();
157     double lvalue;
158     double uvalue;
159     if (dataset instanceof VectorXYDataset) {
160         VectorXYDataset vdataset = (VectorXYDataset) dataset;
161         int series = 0;
162         while (series < seriesCount){
163             int itemCount = dataset.getItemCount(series);
164             int item=0;
165             while (item < itemCount){
166                 double delta = vdataset.getVectorXValue(series, item);
167                 if (delta < 0.0) {
168                     uvalue = vdataset.getXValue(series, item);
169                     lvalue = uvalue + delta;
170                 }
171                 else {
172                     lvalue = vdataset.getXValue(series, item);
173                 }
174                 minimum = Math.min(minimum, lvalue);
175                 item++;
176             }
177             series++;
178         }
179     }
180     else {
181         int series = 0;
182         while (series < seriesCount) {
183             int itemCount = dataset.getItemCount(series);
184             int item = 0;
185             while (item < itemCount){
186                 lvalue = dataset.getXValue(series, item);
187                 minimum = Math.min(minimum, lvalue);
188                 item++;
189             }
190             series++;
191         }
192     }
193     return minimum;
194 }

```

#### Step 4:

Now we take the Co-Slice code and perform another Sliding on {maximum}.

```
1  double maximum = Double.NEGATIVE_INFINITY;
2  int seriesCount = dataset.getSeriesCount();
3  double lvalue;
4  double uvalue;
5  if (dataset instanceof VectorXYDataset) {
6      VectorXYDataset vdataset = (VectorXYDataset) dataset;
7      int series = 0;
8      while (series < seriesCount){
9          int itemCount = dataset.getItemCount(series);
10         int item=0;
11         while (item < itemCount){
12             double delta = vdataset.getVectorXValue(series, item);
13             if (delta < 0.0) {
14                 uvalue = vdataset.getXValue(series, item);
15             }
16             else {
17                 lvalue = vdataset.getXValue(series, item);
18                 uvalue = lvalue + delta;
19             }
20             maximum = Math.max(maximum, uvalue);
21             item++;
22         }
23         series++;
24     }
25 }
26 else {
27     int series = 0;
28     while (series < seriesCount) {
29         int itemCount = dataset.getItemCount(series);
30         int item = 0;
31         while (item < itemCount){
32             lvalue = dataset.getXValue(series, item);
33             uvalue = lvalue;
34             maximum = Math.max(maximum, uvalue);
35             item++;
36         }
37         series++;
38     }
39 }
```

After we apply Sliding on {maximum} we get:

Slice(exit) = {entry, 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 20, 21, 23, 27, 28,29, 30, 31, 32, 33, 34, 35, 37, exit}

Co-Slice(exit) = {}

pen1 = {}

pen2 = {}

pen3 = {}.

### Step 5:

Extract the Slice into a new method.

```
95 public Range findDomainBounds(XYDataset dataset) {
96     if (dataset == null) {
97         throw new IllegalArgumentException("Null 'dataset' argument.");
98     }
99     //*****slice*****
100     int seriesCount;
101     double lvalue;
102     double uvalue;
103     double minimum = findMinimum(dataset);
104
105     //*****co-slice*****
106     double maximum = findMaximum(dataset);
107
108
109     //end of changes
110     if (minimum > maximum) {
111         return null;
112     }
113     else {
114         return new Range(minimum, maximum);
115     }
116 }
117
```

### Step 6:

dead code elimination for lines 100-103 ,since there are declarations for variables that already exists in the extracted methods and now they unnecessary.

```
95 public Range findDomainBounds(XYDataset dataset) {
96     if (dataset == null) {
97         throw new IllegalArgumentException("Null 'dataset' argument.");
98     }
99     double minimum = findMinimum(dataset);
100     double maximum = findMaximum(dataset);
101     if (minimum > maximum) {
102         return null;
103     }
104     else {
105         return new Range(minimum, maximum);
106     }
107 }
108
```

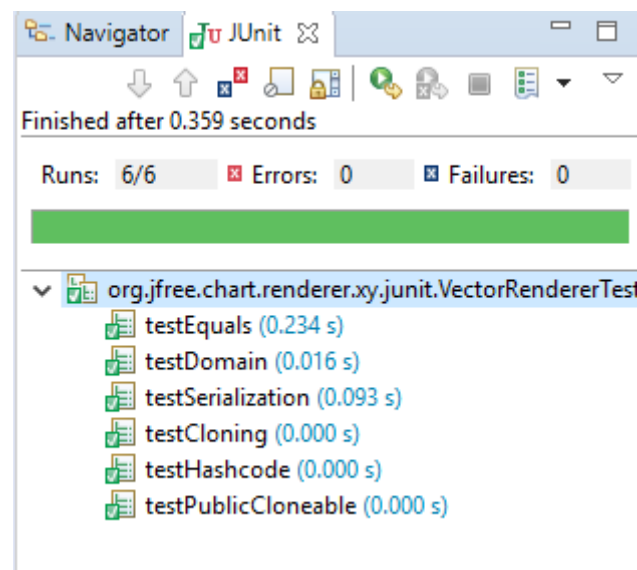
```

109 private double findMaximum(XYDataset dataset) {
110     int seriesCount;
111     double lvalue;
112     double uvalue;
113     double maximum = Double.NEGATIVE_INFINITY;
114     seriesCount = dataset.getSeriesCount();
115     if (dataset instanceof VectorXYDataset) {
116         VectorXYDataset vdataset = (VectorXYDataset) dataset;
117         int series = 0;
118         while (series < seriesCount){
119             int itemCount = dataset.getItemCount(series);
120             int item=0;
121             while (item < itemCount){
122                 double delta = vdataset.getVectorXValue(series, item);
123                 if (delta < 0.0) {
124                     uvalue = vdataset.getXValue(series, item);
125                 }
126                 else {
127                     lvalue = vdataset.getXValue(series, item);
128                     uvalue = lvalue + delta;
129                 }
130                 maximum = Math.max(maximum, uvalue);
131                 item++;
132             }
133             series++;
134         }
135     }
136     else {
137         int series = 0;
138         while (series < seriesCount) {
139             int itemCount = dataset.getItemCount(series);
140             int item = 0;
141             while (item < itemCount){
142                 lvalue = dataset.getXValue(series, item);
143                 uvalue = lvalue;
144                 maximum = Math.max(maximum, uvalue);
145                 item++;
146             }
147             series++;
148         }
149     }
150     return maximum;
151 }
152
153 private double findMinimum(XYDataset dataset) {
154     double minimum = Double.POSITIVE_INFINITY;
155     int seriesCount = dataset.getSeriesCount();
156     double lvalue;
157     double uvalue;
158     if (dataset instanceof VectorXYDataset) {
159         VectorXYDataset vdataset = (VectorXYDataset) dataset;
160         int series = 0;
161         while (series < seriesCount){
162             int itemCount = dataset.getItemCount(series);
163             int item=0;
164             while (item < itemCount){
165                 double delta = vdataset.getVectorXValue(series, item);
166                 if (delta < 0.0) {
167                     uvalue = vdataset.getXValue(series, item);
168                     lvalue = uvalue + delta;
169                 }
170                 else {
171                     lvalue = vdataset.getXValue(series, item);
172                 }
173                 minimum = Math.min(minimum, lvalue);
174                 item++;
175             }
176             series++;
177         }
178     }
179     else {
180         int series = 0;
181         while (series < seriesCount) {
182             int itemCount = dataset.getItemCount(series);
183             int item = 0;
184             while (item < itemCount){
185                 lvalue = dataset.getXValue(series, item);
186                 minimum = Math.min(minimum, lvalue);
187                 item++;
188             }
189             series++;
190         }
191     }
192     return minimum;
193 }

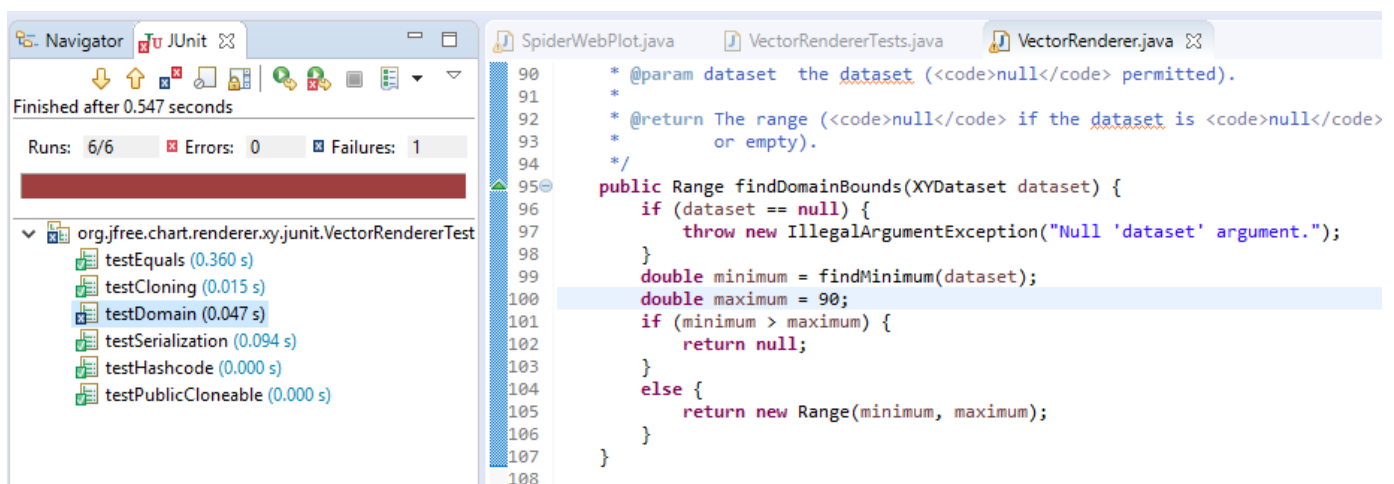
```



Let's test the code:



Let's inject a piece of code that will cause a failure due to intentional mistake.



## Step 7:

While working on the findDomainBounds method, we noticed that another method called findRangeBounds is almost identical.

```
1 public Range findDomainBounds(XYDataset dataset) {
2     if (dataset == null) {
3         throw new IllegalArgumentException("Null 'dataset' argument.");
4     }
5     double minimum = Double.POSITIVE_INFINITY;
6     double maximum = Double.NEGATIVE_INFINITY;
7     int seriesCount = dataset.getSeriesCount();
8     double lvalue;
9     double uvalue;
10    if (dataset instanceof VectorXYDataset) {
11        VectorXYDataset vdataset = (VectorXYDataset) dataset;
12        for (int series = 0; series < seriesCount; series++) {
13            int itemCount = dataset.getItemCount(series);
14            for (int item = 0; item < itemCount; item++) {
15                double delta = vdataset.getVectorXValue(series, item);
16                if (delta < 0.0) {
17                    uvalue = vdataset.getXValue(series, item);
18                    lvalue = uvalue + delta;
19                }
20                else {
21                    lvalue = vdataset.getXValue(series, item);
22                    uvalue = lvalue + delta;
23                }
24                minimum = Math.min(minimum, lvalue);
25                maximum = Math.max(maximum, uvalue);
26            }
27        }
28    }
29    else {
30        for (int series = 0; series < seriesCount; series++) {
31            int itemCount = dataset.getItemCount(series);
32            for (int item = 0; item < itemCount; item++) {
33                lvalue = dataset.getXValue(series, item);
34                uvalue = lvalue;
35                minimum = Math.min(minimum, lvalue);
36                maximum = Math.max(maximum, uvalue);
37            }
38        }
39    }
40    if (minimum > maximum) {
41        return null;
42    }
43    else {
44        return new Range(minimum, maximum);
45    }
46 }
```

```
1 public Range findRangeBounds(XYDataset dataset) {
2     if (dataset == null) {
3         throw new IllegalArgumentException("Null 'dataset' argument.");
4     }
5     double minimum = Double.POSITIVE_INFINITY;
6     double maximum = Double.NEGATIVE_INFINITY;
7     int seriesCount = dataset.getSeriesCount();
8     double lvalue;
9     double uvalue;
10    if (dataset instanceof VectorXYDataset) {
11        VectorXYDataset vdataset = (VectorXYDataset) dataset;
12        for (int series = 0; series < seriesCount; series++) {
13            int itemCount = dataset.getItemCount(series);
14            for (int item = 0; item < itemCount; item++) {
15                double delta = vdataset.getVectorYValue(series, item);
16                if (delta < 0.0) {
17                    uvalue = vdataset.getYValue(series, item);
18                    lvalue = uvalue + delta;
19                }
20                else {
21                    lvalue = vdataset.getYValue(series, item);
22                    uvalue = lvalue + delta;
23                }
24                minimum = Math.min(minimum, lvalue);
25                maximum = Math.max(maximum, uvalue);
26            }
27        }
28    }
29    else {
30        for (int series = 0; series < seriesCount; series++) {
31            int itemCount = dataset.getItemCount(series);
32            for (int item = 0; item < itemCount; item++) {
33                lvalue = dataset.getYValue(series, item);
34                uvalue = lvalue;
35                minimum = Math.min(minimum, lvalue);
36                maximum = Math.max(maximum, uvalue);
37            }
38        }
39    }
40    if (minimum > maximum) {
41        return null;
42    }
43    else {
44        return new Range(minimum, maximum);
45    }
46 }
```

if we do another Sliding to findRangeBound method, we will get the following code:

```
206 public Range findRangeBounds(XYDataset dataset) {
207     if (dataset == null) {
208         throw new IllegalArgumentException("Null 'dataset' argument.");
209     }
210     double minimum = findMinimum2(dataset);
211     double maximum = findMaximum2(dataset);
212
213     if (minimum > maximum) {
214         return null;
215     }
216     else {
217         return new Range(minimum, maximum);
218     }
219 }
```

\*findMinimum2 and findMaximum2 are almost identical to findMinimum and findMaximum, the difference is that findMinimum and findMaximum uses `getVectorXValue` and `getXValue` while the findMinimum2 and findMaximum2 uses `getVectorYValue` and `getYValue`.

Our suggestion is to create another three helper methods as follows:

```
1 //instead of 2 method getXValue and getYValue --> one method getValue
2 //bool==0 -->x -->for domain method, bool==1-->y -->for range method
3 private double getValue(VectorXYDataset vdataset,int series, int item,int bool) {
4
5     if(bool==0){
6         return vdataset.getXValue(series, item);
7     }
8     else {
9         return vdataset.getYValue(series, item);
10    }
11
12 }
13 //instead of 2 method getVectorXValue and getVectorYValue --> one method getVectorValue
14 //bool==0 -->x -->for domain method, bool==1-->y -->for range method
15 private double getVectorValue(VectorXYDataset vdataset,int series, int item,int bool) {
16
17     if(bool==0){
18         return vdataset.getVectorXValue(series, item);
19     }
20     else {
21         return vdataset.getVectorYValue(series, item);
22    }
23
24 }
25 //instead of 2 method getXValue and getYValue --> one method getValue
26 //bool==0 -->x -->for domain method, bool==1-->y -->for range method
27 //this method with XYDataset dataset not VectorXYDataset vdataset --> almost the same
28 private double getDataValue(XYDataset dataset,int series, int item,int bool) {
29
30     if(bool==0){
31         return dataset.getXValue(series, item);
32     }
33     else {
34         return dataset.getYValue(series, item);
35    }
```

now we can merge findMinimum and findMinimum2 to one method, and same for findMaximum and findMaximum2.

Instead calling `getVectorXValue` and `getXValue` and `getVectorYValue` and `getYValue` we can call to the new methods.

In the findDomainBound we will send another argument (0) that represent for the new methods to use `getVectorXValue` and `getXValue`.

In the findRangeBound we will send another argument (1) that represent for the new methods to use `getVectorYValue` and `getYValue`.

\*if we want we can merge findRangeBound and findDomainBound but for that we need to change all the calling to those method (because another argument needed).

The final version will be:

\*we return the loops to the previous conditions, while back to for.

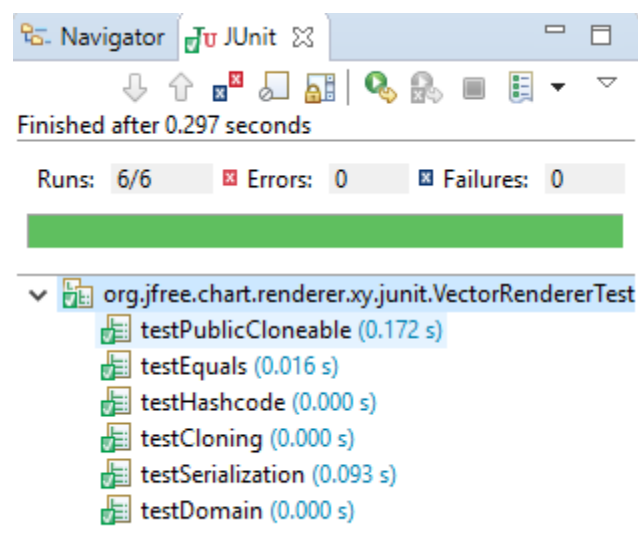
```
95 public Range findDomainBounds(XYDataset dataset) {
96     if (dataset == null) {
97         throw new IllegalArgumentException("Null 'dataset' argument.");
98     }
99     double minimum = findMinimum(dataset,0);
100    double maximum = findMaximum(dataset,0);
101    if (minimum > maximum) {
102        return null;
103    }
104    else {
105        return new Range(minimum, maximum);
106    }
107 }
108 public Range findRangeBounds(XYDataset dataset) {
109     if (dataset == null) {
110         throw new IllegalArgumentException("Null 'dataset' argument.");
111     }
112     double minimum = findMinimum(dataset,1);
113     double maximum = findMaximum(dataset,1);
114     if (minimum > maximum) {
115         return null;
116     }
117     else {
118         return new Range(minimum, maximum);
119     }
120 }
121 private double findMaximum(XYDataset dataset,int domainOrRange) {
122     int seriesCount;
123     double lvalue;
124     double uvalue;
125     double maximum = Double.NEGATIVE_INFINITY;
126     seriesCount = dataset.getSeriesCount();
127     if (dataset instanceof VectorXYDataset) {
128         VectorXYDataset vdataset = (VectorXYDataset) dataset;
129         for (int series = 0; series < seriesCount; series++){
130             int itemCount = dataset.getItemCount(series);
131             for (int item=0; item < itemCount; item++){
132                 double delta = getVectorValue(vdataset,series, item,domainOrRange);
133                 if (delta < 0.0) {
134                     uvalue =getValue(vdataset,series, item,domainOrRange);
135                 }
136                 else {
137                     lvalue = getValue(vdataset,series, item,domainOrRange);
138                     uvalue = lvalue + delta;
139                 }
140                 maximum = Math.max(maximum, uvalue);
141             }
142         }
143     }
144     else {
145         for ( int series = 0; series < seriesCount; series++) {
146             int itemCount = dataset.getItemCount(series);
147             for (int item = 0; item < itemCount; item++){
148                 lvalue = getDataValue(dataset,series, item,domainOrRange);
149                 uvalue = lvalue;
150                 maximum = Math.max(maximum, uvalue);
151             }
152         }
```

```

153     }
154 }
155 }
156 return maximum;
157 }
158 private double findMinimum(XYDataset dataset,int domainOrRange) {
159     double minimum = Double.POSITIVE_INFINITY;
160     int seriesCount = dataset.getSeriesCount();
161     double lvalue;
162     double uvalue;
163     if (dataset instanceof VectorXYDataset) {
164         VectorXYDataset vdataset = (VectorXYDataset) dataset;
165         for (int series = 0; series < seriesCount; series++){
166             int itemCount = dataset.getItemCount(series);
167             for (int item=0; item < itemCount; item++){
168                 double delta =getVectorValue(vdataset,series, item,domainOrRange);
169                 if (delta < 0.0) {
170                     uvalue = getValue(vdataset,series, item,domainOrRange);
171                     lvalue = uvalue + delta;
172                 }
173                 else {
174                     lvalue =getValue(vdataset,series, item,domainOrRange);
175                 }
176                 minimum = Math.min(minimum, lvalue);
177             }
178         }
179     }
180     else {
181         for (int series = 0; series < seriesCount; series++) {
182             int itemCount = dataset.getItemCount(series);
183             int item = 0;
184             while (item < itemCount){
185                 lvalue =getDataValue(dataset,series, item,domainOrRange);
186                 minimum = Math.min(minimum, lvalue);
187                 item++;
188             }
189         }
190     }
191     return minimum;
192 }

```

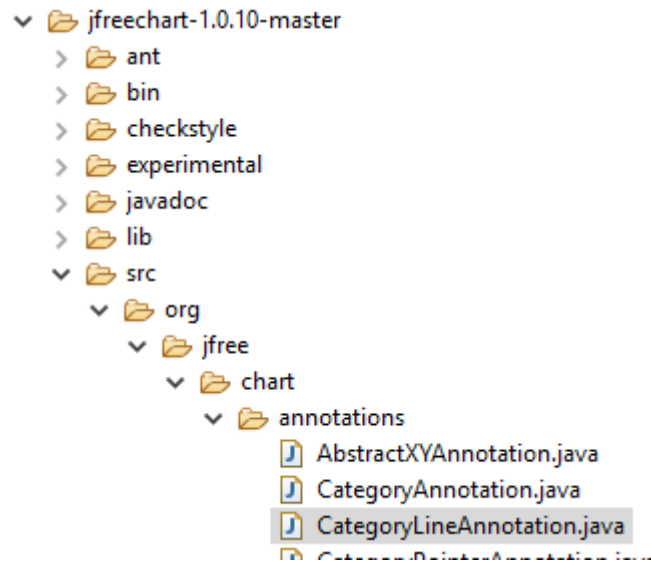
Let's test the code:



### **Bucketing:**

We chose a method called "draw" from an open source project which can be found at

<https://github.com/alwaqfi/jfreechart-1.0.10/blob/master/src/org/jfree/chart/annotations/CategoryLineAnnotation.java>



Since the method is too large, we focused on a specific block of code inside it, and applied the bucketing algorithm on that specific block of code.

We noticed that there are definitions of many variables {lineX1, lineX2, lineY1, lineY2} at a time.

We want the method to be easier to read and maintain by extracting their computations to external methods.

Another reason to perform this extraction is the repeatedly computations for these variables in another methods in this project, by doing so, we can reuse this external methods and avoid repeated code (clone elimination).

### Original code:

```
285 public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
286                  CategoryAxis domainAxis, ValueAxis rangeAxis) {
287
288     CategoryDataset dataset = plot.getDataset();
289     int catIndex1 = dataset.getColumnIndex(this.category1);
290     int catIndex2 = dataset.getColumnIndex(this.category2);
291     int catCount = dataset.getColumnCount();
292
293     double lineX1 = 0.0f;
294     double lineY1 = 0.0f;
295     double lineX2 = 0.0f;
296     double lineY2 = 0.0f;
297
298     PlotOrientation orientation = plot.getOrientation();
299     RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
300         plot.getDomainAxisLocation(), orientation);
301     RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
302         plot.getRangeAxisLocation(), orientation);
303
304     if (orientation == PlotOrientation.HORIZONTAL) {
305         lineY1 = domainAxis.getCategoryJava2DCoordinate(
306             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
307                 domainEdge);
308         lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
309         lineY2 = domainAxis.getCategoryJava2DCoordinate(
310             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
311                 domainEdge);
312         lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
313     }
314     else if (orientation == PlotOrientation.VERTICAL) {
315         lineX1 = domainAxis.getCategoryJava2DCoordinate(
316             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
317                 domainEdge);
318         lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
319         lineX2 = domainAxis.getCategoryJava2DCoordinate(
320             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
321                 domainEdge);
322         lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
323     }
324     g2.setPaint(this.paint);
325     g2.setStroke(this.stroke);
326     g2.drawLine((int) lineX1, (int) lineY1, (int) lineX2, (int) lineY2);
327 }
```

Runs: 5/5    Errors: 0    Failures: 0

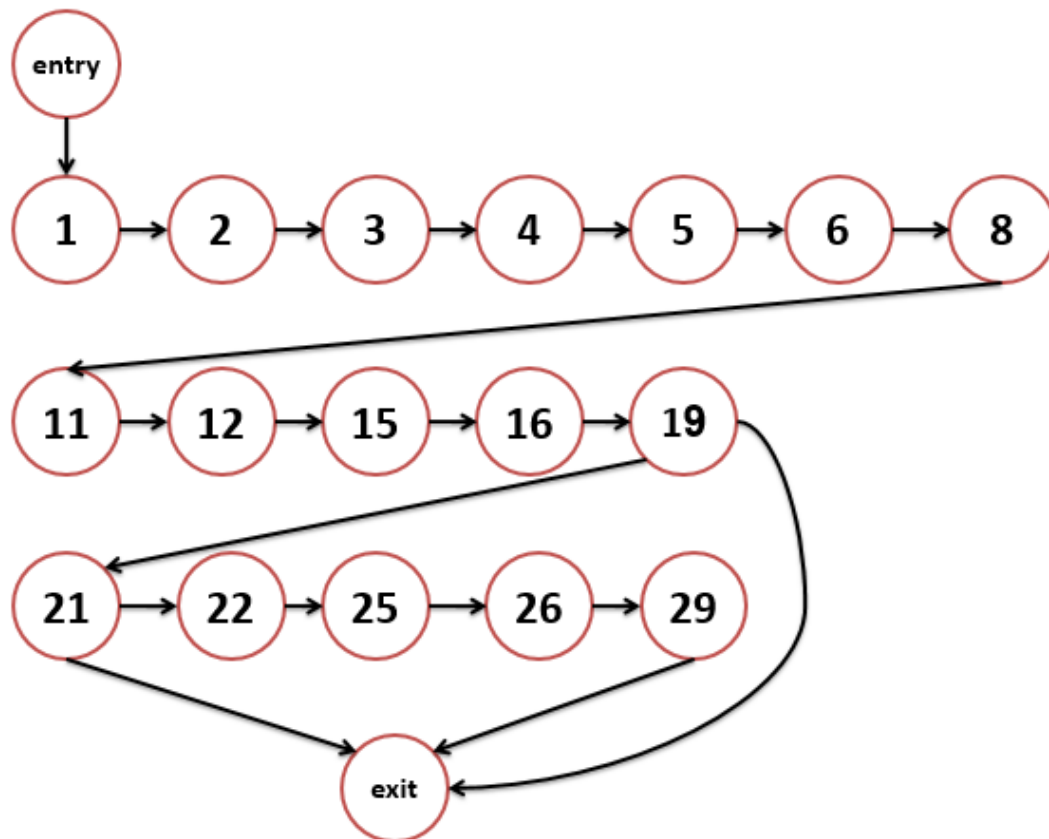
org.jfree.chart.annotations.junit.CategoryLineAnnotationTests

- testSerialization (1.113 s)
- testPublicCloneable (0.000 s)
- testHashCode (0.047 s)
- testEquals (0.000 s)
- testCloning (0.000 s)

We chose the block of code from lines 293-323, for convenience we numbered those lines from 1-25 .

```
1      double lineY1 = 0.0f;
2      double lineX2 = 0.0f;
3      double lineY2 = 0.0f;
4      PlotOrientation orientation = plot.getOrientation();
5      RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
6          plot.getDomainAxisLocation(), orientation);
7      RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
8          plot.getRangeAxisLocation(), orientation);
9
10     if (orientation == PlotOrientation.HORIZONTAL) {
11         lineY1 = domainAxis.getCategoryJava2DCoordinate(
12             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
13             domainEdge);
14         lineY2 = domainAxis.getCategoryJava2DCoordinate(
15             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
16             domainEdge);
17         lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
18     }
19     else if (orientation == PlotOrientation.VERTICAL) {
20         lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
21         lineX2 = domainAxis.getCategoryJava2DCoordinate(
22             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
23             domainEdge);
24         lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
25     }
```

Build the CFG:





### Build the PDG:

Before we can perform the bucketing, we need to compute the PDG (we did it with {entry, exit} nodes for the correctness of the PDG, although we won't need them for the bucketing).

Edge	type
(Entry,1)	<b>Control</b>
(Entry,2)	<b>Control</b>
(Entry,3)	<b>Control</b>
(Entry,4)	<b>Control</b>
(Entry,5)	<b>Control</b>
(Entry,6)	<b>Control</b>
(Entry,8)	<b>Control</b>
(Entry,11)	<b>Control</b>
(Entry,21)	<b>Control</b>
(11,12)	<b>Control</b>
(11,15)	<b>Control</b>
(11,16)	<b>Control</b>
(11,19)	<b>Control</b>
(21,22)	<b>Control</b>
(21,25)	<b>Control</b>
(21,26)	<b>Control</b>
(21,29)	<b>Control</b>
(29, exit)	<b>Control</b>
(19, exit)	<b>Control</b>

Edge	type	Vars
(1,15)	output	{lineX1}
(1,22)	output	{lineX1}
(2,12)	output	{lineY1}
(2,25)	output	{lineY1}
(3,19)	output	{lineX2}
(3,26)	output	{lineX2}
(4,16)	output	{lineY2}
(4,29)	output	{lineY2}

Edge	type	Vars
(5,6)	Flow	{ orientation }
(5,8)	Flow	{ orientation }
(5,11)	Flow	{ orientation }
(5,21)	Flow	{ orientation }
(entry,5)	Flow	{ plot }
(entry,6)	Flow	{ plot }
(entry,8)	Flow	{ plot }
(6,12)	Flow	{ domainEdge }
(6,16)	Flow	{ domainEdge }
(6,22)	Flow	{ domainEdge }
(6,26)	Flow	{ domainEdge }
(8,15)	Flow	{ rangeEdge }
(8,19)	Flow	{ rangeEdge }
(8,25)	Flow	{ rangeEdge }
(8,29)	Flow	{ rangeEdge }
(entry,12)	Flow	{ domainAxis }
(entry,16)	Flow	{ domainAxis }
(entry,22)	Flow	{ domainAxis }
(entry,26)	Flow	{ domainAxis }
(entry,15)	Flow	{ rangeAxis }
(entry,19)	Flow	{ rangeAxis }
(entry,25)	Flow	{ rangeAxis }
(entry,29)	Flow	{ rangeAxis }
(entry,12)	Flow	{ catIndex1 }
(entry,22)	Flow	{ catIndex1 }
(entry,16)	Flow	{ catIndex2 }
(entry,26)	Flow	{ catIndex2 }

Edge	type	Vars
(entry,12)	Flow	{ catCount }
(entry,16)	Flow	{ catCount }
(entry,22)	Flow	{ catCount }
(entry,26)	Flow	{ catCount }
(entry,12)	Flow	{ dataArea }
(entry,15)	Flow	{ dataArea }
(entry,16)	Flow	{ dataArea }
(entry,19)	Flow	{ dataArea }
(entry,22)	Flow	{ dataArea }
(entry,25)	Flow	{ dataArea }
(entry,26)	Flow	{ dataArea }
(entry,29)	Flow	{ dataArea }
(entry,22)	Flow	{ this.value1 }
(entry,15)	Flow	{ this.value1 }
(entry,25)	Flow	{ this.value1 }
(entry,29)	Flow	{ this.value2 }
(entry,19)	Flow	{ this.value2 }
(15,exit)	Flow	{ lineX1 }
(22,exit)	Flow	{ lineX1 }
(12,exit)	Flow	{ lineY1 }
(25,exit)	Flow	{ lineY1 }
(19,exit)	Flow	{ lineX2 }
(26,exit)	Flow	{ lineX2 }
(16,exit)	Flow	{ lineY2 }
(29,exit)	Flow	{ lineY2 }

### Build the Slide-DG:

For building the Slide-DG, we will remove all the predicate nodes and {entry ,exit} and their edges.

We will copy all the other nodes and their edges to a new graph and add the slide dependence to the graph.

For computing the slide dependence, we first need to compute the slide for each node.

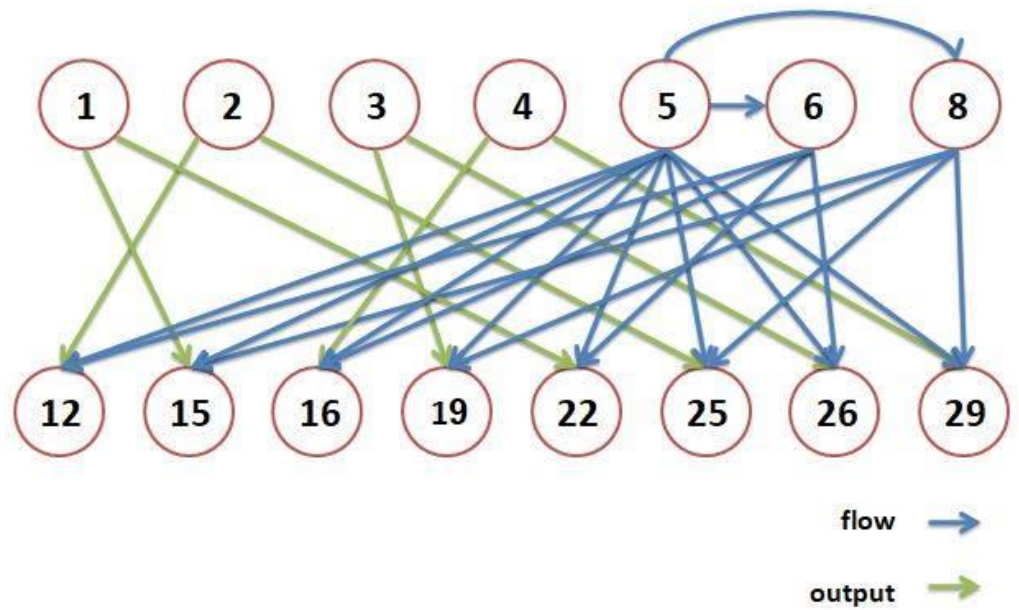
slide(1)= {1}	slide(12)= {12,11}	slide(15)= {15,11}
slide(2)= {2}	slide(16)= {16,11}	slide(19)= {19,11}
slide(3)= {3}	slide(22)= {22,21}	slide(25)= {25,21}
slide(4)= {4}	slide(26)= {26,21}	slide(29)= {29,21}
slide(5)= {5}	slide(6)= {6}	slide(8)= {8}

Now we get this new slide dependence edges:

Edge
(5,12)
(5,19)
(5,16)
(5,15)
(5,22)
(5,26)
(5,29)
(5,25)

After those steps we get this Slide-DG:

Edge
(5,6)
(5,8)
(6,12)
(6,16)
(6,22)
(6,26)
(8,15)
(8,19)
(8,25)
(8,29)
(1,15)
(1,22)
(2,12)
(2,25)
(3,19)
(3,26)
(4,16)
(4,29)
(5,12)
(5,19)
(5,16)
(5,15)
(5,22)
(5,26)
(5,29)
(5,25)



Our purpose is to extract each variable computation to external method, we noticed that the order of the extractions is arbitrary, we chose to apply the bucketing algorithm on "lineX1".

For that purpose we chose  $M = \{1,15,22\}$ , since those lines are the definitions to "lineX1".

**Reaching -M** =  $\{1,15,22,5,6,8\}$

**M -Reachable** =  $\{1,15,22\}$

Buckets are computed as follows:

- Before =  $\{\text{Reaching-M} \setminus \text{M-Reachable}\}$
- After =  $\{\text{M-Reachable} \setminus \text{Reaching-M}\}$
- Marked =  $\{\text{Reaching-M} \cap \text{M-Reachable}\}$  Predicates corresponding to the nodes of each bucket are added as well.

Therefore:

Before =  $\{6,5,8\}$

After =  $\{\}$

Marked =  $\{1,15,22\}$

Result after first step:

```
1 //before
2 PlotOrientation orientation = plot.getOrientation();
3 RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
4     plot.getDomainAxisLocation(), orientation);
5 RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
6     plot.getRangeAxisLocation(), orientation);
7
8 //marked
9 double lineX1 = 0.0f;
10 ▼ if (orientation == PlotOrientation.HORIZONTAL) {
11     lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
12 }
13 ▼ else if (orientation == PlotOrientation.VERTICAL) {
14 ▼     lineX1 = domainAxis.getCategoryJava2DCoordinate(
15         CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
16         domainEdge);
17     }
18 //after --
19 double lineY1 = 0.0f;
20 double lineX2 = 0.0f;
21 double lineY2 = 0.0f;
22 ▼ if (orientation == PlotOrientation.HORIZONTAL) {
23 ▼     lineY1 = domainAxis.getCategoryJava2DCoordinate(
24         CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
25         domainEdge);
26 ▼     lineY2 = domainAxis.getCategoryJava2DCoordinate(
27         CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
28         domainEdge);
29     lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
30 }
31 else if (orientation == PlotOrientation.VERTICAL) {
32     lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
33     lineX2 = domainAxis.getCategoryJava2DCoordinate(
34         CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
35         domainEdge);
36     lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
37 }
```

After applying the bucketing, we noticed that lines 2-6 will always appear in the "before" since we need those lines to define those variables.

Since lines 19-37 are arbitrary, they will be in the "after" and in the following steps we will apply the bucketing only on those lines.

The marked bucket will be extracted to an external method called computeLineX1.

The result will be:

```
286 public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
287                  CategoryAxis domainAxis, ValueAxis rangeAxis) {
288
289     CategoryDataset dataset = plot.getDataset();
290     int catIndex1 = dataset.getColumnIndex(this.category1);
291     int catIndex2 = dataset.getColumnIndex(this.category2);
292     int catCount = dataset.getColumnCount();
293     //before
294     PlotOrientation orientation = plot.getOrientation();
295     RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
296         plot.getDomainAxisLocation(), orientation);
297     RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
298         plot.getRangeAxisLocation(), orientation);
299
300     //marked ---->will do extract to the marked part
301     double lineX1 = computeLineX1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
302         rangeEdge);
303     //after --
304     double lineY1 = 0.0f;
305     double lineX2 = 0.0f;
306     double lineY2 = 0.0f;
307     if (orientation == PlotOrientation.HORIZONTAL) {
308         lineY1 = domainAxis.getCategoryJava2DCoordinate(
309             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
310             domainEdge);
311         lineY2 = domainAxis.getCategoryJava2DCoordinate(
312             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
313             domainEdge);
314         lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
315     }
316     else if (orientation == PlotOrientation.VERTICAL) {
317         lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
318         lineX2 = domainAxis.getCategoryJava2DCoordinate(
319             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
320             domainEdge);
321         lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
322     }
323
324     g2.setPaint(this.paint);
325     g2.setStroke(this.stroke);
326     g2.drawLine((int) lineX1, (int) lineY1, (int) lineX2, (int) lineY2);
327 }
328
329 private double computeLineX1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
330                             int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
331     double lineX1 = 0.0f;
332     if (orientation == PlotOrientation.HORIZONTAL) {
333         lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
334     }
335     else if (orientation == PlotOrientation.VERTICAL) {
336         lineX1 = domainAxis.getCategoryJava2DCoordinate(
337             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
338             domainEdge);
339     }
340     return lineX1;
341 }
```

Runs: 5/5   Errors: 0   Failures: 0

org.jfree.chart.annotations.junit.CategoryLineAnnotationTests

- testSerialization (0.144 s)
- testPublicCloneable (0.001 s)
- testEquals (0.000 s)
- testCloning (0.000 s)
- testHashCode (0.001 s)

## Step 2:

we apply the bucketing algorithm on lineY1.

```
1 double lineY1 = 0.0f;
2 double lineX2 = 0.0f;
3 double lineY2 = 0.0f;
4 if (orientation == PlotOrientation.HORIZONTAL) {
5     lineY1 = domainAxis.getCategoryJava2DCoordinate(
6         CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
7         domainEdge);
8     lineY2 = domainAxis.getCategoryJava2DCoordinate(
9         CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
10        domainEdge);
11    lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
12 }
13 else if (orientation == PlotOrientation.VERTICAL) {
14    lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
15    lineX2 = domainAxis.getCategoryJava2DCoordinate(
16        CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
17        domainEdge);
18    lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
19 }
```

$M = \{1,5,14\}$

the result after step 2:

```
286 public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
287                  CategoryAxis domainAxis, ValueAxis rangeAxis) {
288
289     CategoryDataset dataset = plot.getDataset();
290     int catIndex1 = dataset.getColumnIndex(this.category1);
291     int catIndex2 = dataset.getColumnIndex(this.category2);
292     int catCount = dataset.getColumnCount();
293
294     PlotOrientation orientation = plot.getOrientation();
295     RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
296         plot.getDomainAxisLocation(), orientation);
297     RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
298         plot.getRangeAxisLocation(), orientation);
299
300     double lineX1 = computeLineX1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
301        rangeEdge);
302     double lineY1 = computeLineY1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
303        rangeEdge);
304
305     double lineX2 = 0.0f;
306     double lineY2 = 0.0f;
307     if (orientation == PlotOrientation.HORIZONTAL) {
308         lineY2 = domainAxis.getCategoryJava2DCoordinate(
309             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
310             domainEdge);
311         lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
312     }
313     else if (orientation == PlotOrientation.VERTICAL) {
314         lineX2 = domainAxis.getCategoryJava2DCoordinate(
315             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
316             domainEdge);
317         lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
318     }
319
320     g2.setPaint(this.paint);
321     g2.setStroke(this.stroke);
322     g2.drawLine((int) lineX1, (int) lineY1, (int) lineX2, (int) lineY2);
323 }
324
325 private double computeLineX1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
326    int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
327     double lineX1 = 0.0f;
328     if (orientation == PlotOrientation.HORIZONTAL) {
329         lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
330     }
331     else if (orientation == PlotOrientation.VERTICAL) {
332         lineX1 = domainAxis.getCategoryJava2DCoordinate(
333             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
334             domainEdge);
335     }
336     return lineX1;
337 }
338
339 private double computeLineY1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
340    int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
341     double lineY1 = 0.0f;
342     if (orientation == PlotOrientation.HORIZONTAL) {
343         lineY1 = domainAxis.getCategoryJava2DCoordinate(
344             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
345             domainEdge);
346     }
347     else if (orientation == PlotOrientation.VERTICAL) {
348         lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
349     }
350     return lineY1;
351 }
```



In the same way we performed step3 and step4 for the other variables.

The final result will be :

```
285 public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
286     CategoryAxis domainAxis, ValueAxis rangeAxis) {
287
288     CategoryDataset dataset = plot.getDataset();
289     int catIndex1 = dataset.getColumnIndex(this.category1);
290     int catIndex2 = dataset.getColumnIndex(this.category2);
291     int catCount = dataset.getColumnCount();
292
293     PlotOrientation orientation = plot.getOrientation();
294     RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(plot.getDomainAxisLocation(), orientation);
295     RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(plot.getRangeAxisLocation(), orientation);
296
297     double lineX1 = computeLineX1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
298     rangeEdge);
299     double lineY1 = computeLineY1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
300     rangeEdge);
301     double lineX2 = computeLineX2(dataArea, domainAxis, rangeAxis, catIndex2, catCount, orientation, domainEdge,
302     rangeEdge);
303     double lineY2 = computeLineY2(dataArea, domainAxis, rangeAxis, catIndex2, catCount, orientation, domainEdge,
304     rangeEdge);
305
306     g2.setPaint(this.paint);
307     g2.setStroke(this.stroke);
308     g2.drawLine((int) lineX1, (int) lineY1, (int) lineX2, (int) lineY2);
309 }
310
311 private double computeLineX1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
312     int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
313     double lineX1 = 0.0f;
314     if (orientation == PlotOrientation.HORIZONTAL) {
315         lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
316     }
317     else if (orientation == PlotOrientation.VERTICAL) {
318         lineX1 = domainAxis.getCategoryJava2DCoordinate(
319             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea, domainEdge);
320     }
321     return lineX1;
322 }
323
324 private double computeLineY1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
325     int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
326     double lineY1 = 0.0f;
327     if (orientation == PlotOrientation.HORIZONTAL) {
328         lineY1 = domainAxis.getCategoryJava2DCoordinate(
329             CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea, domainEdge);
330     }
331     else if (orientation == PlotOrientation.VERTICAL) {
332         lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
333     }
334     return lineY1;
335 }
336
337 private double computeLineX2(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex2,
338     int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
339     double lineX2 = 0.0f;
340     if (orientation == PlotOrientation.HORIZONTAL) {
341         lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
342     }
343     else if (orientation == PlotOrientation.VERTICAL) {
344         lineX2 = domainAxis.getCategoryJava2DCoordinate(
345             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea, domainEdge);
346     }
347     return lineX2;
348 }
349
350 private double computeLineY2(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex2,
351     int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
352     double lineY2 = 0.0f;
353     if (orientation == PlotOrientation.HORIZONTAL) {
354         lineY2 = domainAxis.getCategoryJava2DCoordinate(
355             CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea, domainEdge);
356     }
357     else if (orientation == PlotOrientation.VERTICAL) {
358         lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
359     }
360     return lineY2;
361 }
362 }
```

**Conclusion:**

In conclusion, the draw method is simpler then before and easier to maintain.

Now we have four helper methods and we can reuse them in other places in the project, although, the runtime might be larger .

Looking back, in this specific code one can argue that it would be better to leave the code as it is because those four variables have the same purpose (coordinates) and should be compute together, nonetheless, we decided to do so for the purpose of learning as well.

## SQFM:

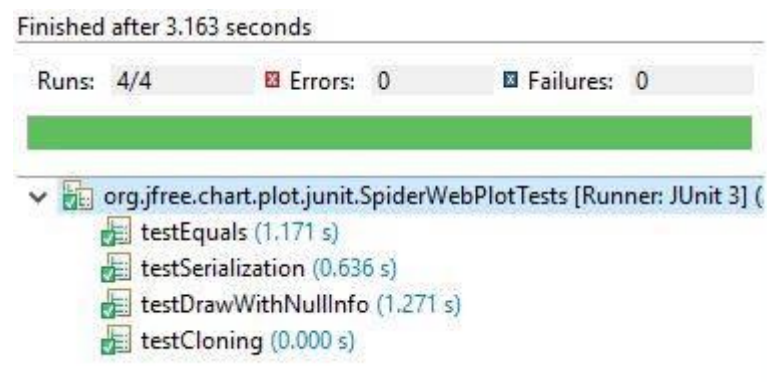
We chose a method called "getSeriesPaint" from an open source project which can be found at:

<https://github.com/alwaqfi/jfreechart-1.0.10/blob/master/src/org/jfree/chart/plot/SpiderWebPlot.java>

[jfreechart-1.0.10/src/org/jfree/chart/plot/SpiderWebPlot.java](https://github.com/alwaqfi/jfreechart-1.0.10/blob/master/src/org/jfree/chart/plot/SpiderWebPlot.java)

Original code:

```
686 public Paint getSeriesPaint(int series) {
687
688     // return the override, if there is one...
689     if (this.seriesPaint != null) {
690         return this.seriesPaint;
691     }
692
693     // otherwise look up the paint list
694     Paint result = this.seriesPaintList.getPaint(series);
695     if (result == null) {
696         DrawingSupplier supplier = getDrawingSupplier();
697         if (supplier != null) {
698             Paint p = supplier.getNextPaint();
699             this.seriesPaintList.setPaint(series, p);
700             result = p;
701         }
702         else {
703             result = this.baseSeriesPaint;
704         }
705     }
706     return result;
707 }
708 }
```



As we can see this is a non-void method that have side effect on object field (line 669: `this.seriesPaintList.setPaint(series,p);` )

Therefore, we apply sperate query from modifier refactoring.

## Step 1:

In this step we prepared the code for Sliding algorithm by adjusting the code to have a single return value

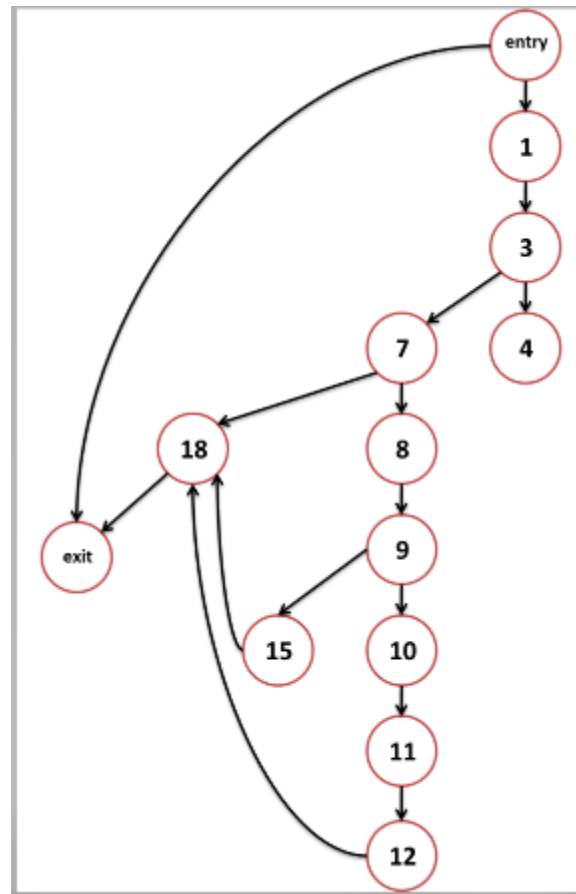
\* this attempt is a wrong one that might cause problems.

```
686 public Paint getSeriesPaint(int series) {
687     Paint result = this.seriesPaintList.getPaint(series);
688     // return the override, if there is one...
689     if (this.seriesPaint != null) {
690         result = this.seriesPaint;
691     }
692     // otherwise look up the paint list
693     if (result == null) {
694         DrawingSupplier supplier = getDrawingSupplier();
695         if (supplier != null) {
696             Paint p = supplier.getNextPaint();
697             this.seriesPaintList.setPaint(series, p);
698             result = p;
699         }
700         else {
701             result = this.baseSeriesPaint;
702         }
703     }
704     return result;
705 }
```

We chose the block of code from lines 687-704, for convenience we numbered those lines from 1-18.

```
1 Paint result = this.seriesPaintList.getPaint(series);
2 // return the override, if there is one...
3 if (this.seriesPaint != null) {
4     result = this.seriesPaint;
5 }
6 // otherwise look up the paint list
7 if (result == null) {
8     DrawingSupplier supplier = getDrawingSupplier();
9     if (supplier != null) {
10         Paint p = supplier.getNextPaint();
11         this.seriesPaintList.setPaint(series, p);
12         result = p;
13     }
14     else {
15         result = this.baseSeriesPaint;
16     }
17 }
18 return result;
```

Build the CFG:



### Build the PDG:

Before we can perform the bucketing, we need to compute the PDG.

\*in case of a non- void method we address the "return line" as a definition to a variable called ret\_val.

\*\*we assume that the method "getNextPaint" change the supplier object thus we add flow dependence (10, exit).

\*\*\* we assume that the method "getPaint" is only a getter and doesn't change anything.

Edge	type
(Entry,1)	<b>Control</b>
(Entry,3)	<b>Control</b>
(Entry,7)	<b>Control</b>
(Entry,18)	<b>Control</b>
(3,4)	<b>Control</b>
(7,8)	<b>Control</b>
(7,9)	<b>Control</b>
(9,10)	<b>Control</b>
(9,11)	<b>Control</b>
(9,12)	<b>Control</b>
(9,15)	<b>Control</b>

Edge	type	Vars
(7,15)	anti	{ result }
(7,12)	anti	{ result }
(11,11)	anti	{ this.seriesPaintList , this.seriesPaintList .* }
(1,11)	anti	{ this.seriesPaintList , this.seriesPaintList .* }
(9,10)	anti	{ supplier }

Edge	type	Vars
(1, Exit)	flow	{result}
(4,7)	flow	{result}
(1,7)	flow	{result}
(1,18)	flow	{result}
(4,18)	flow	{result}
(12,18)	flow	{result}
(15,18)	flow	{result}
(4, Exit)	flow	{result}
(12, Exit)	flow	{result}
(15, Exit)	flow	{result}
(entry,3)	flow	{this.seriesPaint}
(entry,4)	flow	{this.seriesPaint}
(8,9)	flow	{ supplier }
(8,10)	flow	{supplier , supplier.*}
(8,exit)	flow	{supplier}
(10,11)	flow	{p}
(10,12)	flow	{p}
(10,exit)	flow	{p, supplier, supplier.*}
(entry,11)	flow	{ series }
(entry,1)	flow	{ this.seriesPaintList.* ,this.seriesPaintList ,series}
(11,exit)	flow	{ this.seriesPaintList, this.seriesPaintList .* }
(entry,15)	flow	{ this.baseSeriesPaint }
(18,exit)	flow	{ ret_val }

## Step 2:

We apply sliding algorithm on  $V=\{\text{result}\}$ , the Slice will be the Query and the Co-Slice will be the Modifier.

Slice:

To do the slice we remove flow dependencies to exit of any variable other than "result".

Now we do back tracking starting from exit node up, using control and flow dependencies.

$\text{Slice}(\text{exit}) = \{\text{entry}, 1, 3, 4, 7, 8, 9, 10, 12, 15, \text{exit}\}$

Co- Slice:

To do the Co-Slice we remove flow dependence created by the variable "result" (y, x) such that there is no anti dependence (x, z) created by the variable "result".

Those are the edges that left:

Edge	type
(Entry,1)	<b>Control</b>
(Entry,3)	<b>Control</b>
(Entry,7)	<b>Control</b>
(Entry,18)	<b>Control</b>
(3,4)	<b>Control</b>
(7,8)	<b>Control</b>
(7,9)	<b>Control</b>
(9,10)	<b>Control</b>
(9,11)	<b>Control</b>
(9,12)	<b>Control</b>
(9,15)	<b>Control</b>

Edge	type	Vars
(4,7)	flow	<b>{result}</b>
(1,7)	flow	<b>{result}</b>
(entry,3)	flow	<b>{this.seriesPaint}</b>
(entry,4)	flow	<b>{this.seriesPaint}</b>
(8,9)	flow	<b>{ supplier }</b>
(8,10)	flow	<b>{supplier , supplier.*}</b>
(8,exit)	flow	<b>{supplier}</b>
(10,11)	flow	<b>{p}</b>
(10,12)	flow	<b>{p}</b>
(10,exit)	flow	<b>{p, supplier, supplier.*}</b>
(entry,11)	flow	<b>{ this.seriesPaintList.* ,this.seriesPaintList ,series}</b>
(entry,1)	flow	<b>{ this.seriesPaintList.* ,this.seriesPaintList ,series}</b>
(11,exit)	flow	<b>{ this.seriesPaintList, this.seriesPaintList .* }</b>
(entry,15)	flow	<b>{ this.baseSeriesPaint }</b>
(18,exit)	flow	<b>{ ret_val }</b>

Now we do back tracking starting from exit node up, using control and flow dependencies.

$\text{Co-Slice}(\text{exit}) = \{\text{entry}, 1, 3, 4, 7, 8, 9, 10, 11, 18\}$

### Compensations checking:

**Pen1:** The value of the extracted variable could change in the Co-Slice.

We noticed that node 1, 4 ,existing in Co-Slice, define variable "result" therefore, **pen1={1,result}, [4, result]}**.

**Pen2:** there exists a usage in the Co-Slice of a non-final value of the extracted variable.

We noticed that node 7, existing in Co-Slice, uses variable "result" and there is anti-dependence from node 7 therefore, **pen2={7,result]}**.

**Pen3:** the slice could change the value of unextracted variable, that its initial value is required at the co-slice.

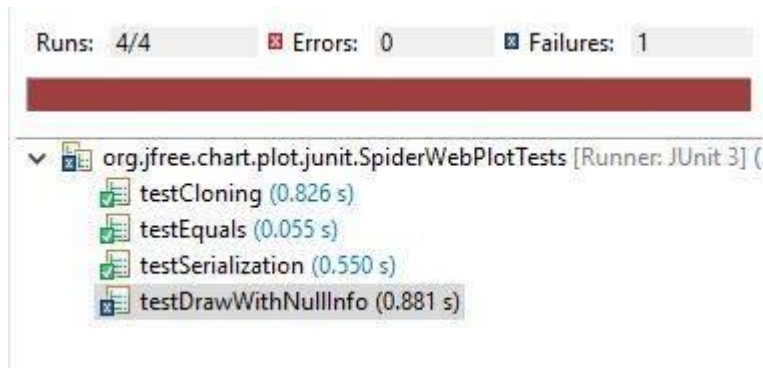
There isn't Slice's nodes defining variables causing flow dependencies from entry to Co-Slice's nodes, therefore **pen3 = {}**.

The result for our first attempt will be:

```
707 public Paint getSeriesPaint(int series) {
708     //slice
709     Paint result = this.seriesPaintList.getPaint(series);
710     if (this.seriesPaint != null) {
711         result= this.seriesPaint;
712     }
713     if (result == null) {
714         DrawingSupplier supplier = getDrawingSupplier();
715         if (supplier != null) {
716             Paint p = supplier.getNextPaint();
717             result = p;
718         }
719         else {
720             result = this.baseSeriesPaint;
721         }
722     }
723
724     //co-slice
725     result = this.seriesPaintList.getPaint(series);
726     // return the override, if there is one...
727     if (this.seriesPaint != null) {
728         result= this.seriesPaint;
729     }
730     if (result == null) {
731         DrawingSupplier supplier = getDrawingSupplier();
732         if (supplier != null) {
733             Paint p = supplier.getNextPaint();
734             this.seriesPaintList.setPaint(series, p);
735         }
736     }
737     return result;
738 }
739
```

As we can see, this result is wrong and not even efficient.





## Second attempt – Sliding:

### Step1:

To correct the problem from the first attempt we will perform this change:

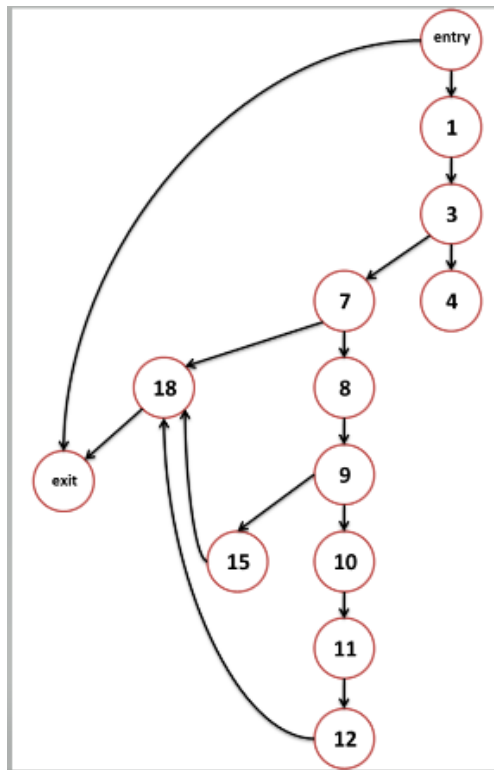
```
686 public Paint getSeriesPaint(int series) {
687
688     Paint result = this.seriesPaintList.getPaint(series);
689     if (this.seriesPaint != null) {
690         result = this.seriesPaint;
691     }
692
693     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
694         DrawingSupplier supplier = getDrawingSupplier();
695         if (supplier != null) {
696             Paint p = supplier.getNextPaint();
697             this.seriesPaintList.setPaint(series, p);
698             result = p;
699         }
700         else {
701             result = this.baseSeriesPaint;
702         }
703     }
704     return result;
705 }
706 }
```

This change will keep the correctness of the code and now we do the Sliding algorithm again.

We chose the block of code from lines 688-704, for convenience we numbered those lines from 1-18.

```
1 Paint result = this.seriesPaintList.getPaint(series);
2
3 if (this.seriesPaint != null) {
4     result = this.seriesPaint;
5 }
6
7 if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
8     DrawingSupplier supplier = getDrawingSupplier();
9     if (supplier != null) {
10         Paint p = supplier.getNextPaint();
11         this.seriesPaintList.setPaint(series, p);
12         result = p;
13     }
14     else {
15         result = this.baseSeriesPaint;
16     }
17 }
18 return result;
```

**Build the CFG:**



### Build the PDG:

Edge	type
(Entry,1)	<b>Control</b>
(Entry,3)	<b>Control</b>
(Entry,7)	<b>Control</b>
(Entry,18)	<b>Control</b>
(3,4)	<b>Control</b>
(7,8)	<b>Control</b>
(7,9)	<b>Control</b>
(9,10)	<b>Control</b>
(9,11)	<b>Control</b>
(9,12)	<b>Control</b>
(9,15)	<b>Control</b>

Edge	type	Vars
(11,11)	anti	<b>{ this.seriesPaintList , this.seriesPaintList . * }</b>
(1,11)	anti	<b>{ this.seriesPaintList , this.seriesPaintList . * }</b>
(9,10)	anti	<b>{ supplier }</b>

Edge	type	Vars
(1,Exit)	flow	<b>{result}</b>
(1,18)	flow	<b>{result}</b>
(4,18)	flow	<b>{result}</b>
(12,18)	flow	<b>{result}</b>
(15,18)	flow	<b>{result}</b>
(4,Exit)	flow	<b>{result}</b>
(12,Exit)	flow	<b>{result}</b>
(15,Exit)	flow	<b>{result}</b>
(entry,3)	flow	<b>{this.seriesPaint}</b>
(entry,4)	flow	<b>{this.seriesPaint}</b>
(entry,7)	flow	<b>{this.seriesPaint}</b>
(8,9)	flow	<b>{ supplier }</b>
(8,10)	flow	<b>{supplier , supplier.*}</b>
(8,exit)	flow	<b>{supplier}</b>
(10,11)	flow	<b>{p}</b>
(10,12)	flow	<b>{p}</b>
(10,exit)	flow	<b>{p, supplier, supplier.*}</b>
(entry,11)	flow	<b>{ this.seriesPaintList.* ,this.seriesPaintList ,series}</b>
(entry,1)	flow	<b>{ this.seriesPaintList.* ,this.seriesPaintList ,series}</b>
(entry,7)	flow	<b>{ this.seriesPaintList.* ,this.seriesPaintList ,series}</b>
(11,exit)	flow	<b>{ this.seriesPaintList, this.seriesPaintList . * }</b>
(entry,15)	flow	<b>{ this.baseSeriesPaint }</b>
(18,exit)	flow	<b>{ ret_val }</b>

Slice = {entry, 1, 3, 4, 7, 8, 9, 10, 12, 15, exit}

Co-Slice = {entry, 7, 8, 9, 10, 11, 18, exit}

## Compensations checking:

**Pen1:** The value of the extracted variable could change in the Co-Slice.

There is no definition to variable "result" in any Co-Slice's nodes, therefore **pen1 = {}**.

**Pen2:** there exists a usage in the Co-Slice of a non-final value of the extracted variable.

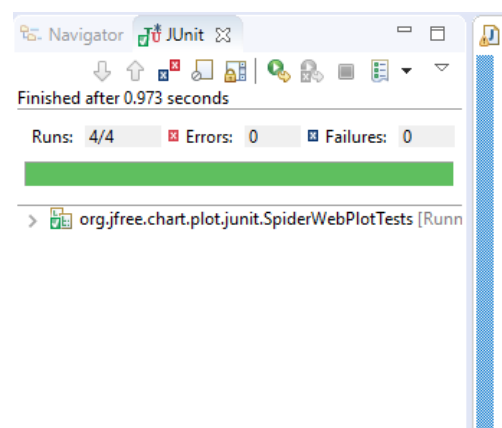
There is no usage to variable "result" in any Co-Slice's nodes, therefore **pen2 = {}**.

**Pen3:** the slice could change the value of unextracted variable, that its initial value is required at the co-slice.

There isn't Slice's nodes defining variables causing flow dependencies from entry to Co-Slice's nodes, therefore **pen3 = {}**.

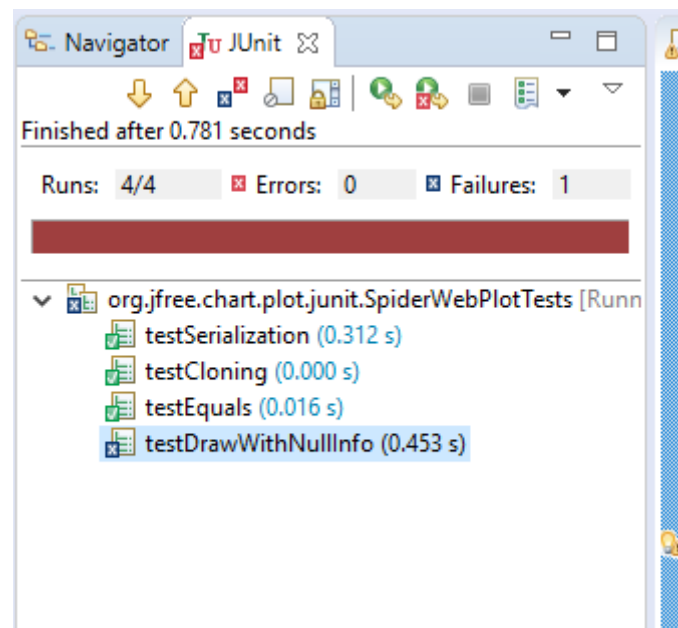
The result will be:

```
686 public Paint getSeriesPaint(int series) {
687     //Slice
688     Paint result = this.seriesPaintList.getPaint(series);
689     if (this.seriesPaint != null) {
690         result = this.seriesPaint;
691     }
692     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
693         DrawingSupplier supplier = getDrawingSupplier();
694         if (supplier != null) {
695             Paint p = supplier.getNextPaint();
696             result = p;
697         }
698         else {
699             result = this.baseSeriesPaint;
700         }
701     }
702     //Co-Slice
703     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
704         DrawingSupplier supplier = getDrawingSupplier();
705         if (supplier != null) {
706             Paint p = supplier.getNextPaint();
707             this.seriesPaintList.setPaint(series, p);
708         }
709     }
710     return result;
711 }
```



Let's inject a piece of code that will cause a failure due to intentional mistake.

```
686 public Paint getSeriesPaint(int series) {
687     //Slice
688     Paint result = this.seriesPaintList.getPaint(series);
689     if (this.seriesPaint != null) {
690         result = this.seriesPaint;
691     }
692     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
693         DrawingSupplier supplier = getDrawingSupplier();
694         if (supplier != null) {
695             Paint p = supplier.getNextPaint();
696             result = null;
697         }
698         else {
699             result = this.baseSeriesPaint;
700         }
701     }
702     //Co-Slice
703     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
704         DrawingSupplier supplier = getDrawingSupplier();
705         if (supplier != null) {
706             Paint p = supplier.getNextPaint();
707             this.seriesPaintList.setPaint(series, p);
708         }
709     }
710     return result;
711 }
```



### Step3:

Extract method to Query.

```
686 public Paint getSeriesPaint(int series) {
687     //Slice
688     Paint result = computeResult(series);
689     //Co-Slice
690     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
691         DrawingSupplier supplier = getDrawingSupplier();
692         if (supplier != null) {
693             Paint p = supplier.getNextPaint();
694             this.seriesPaintList.setPaint(series, p);
695         }
696     }
697     return result;
698 }
699
700 private Paint computeResult(int series) {
701     Paint result = this.seriesPaintList.getPaint(series);
702     if (this.seriesPaint != null) {
703         result = this.seriesPaint;
704     }
705     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
706         DrawingSupplier supplier = getDrawingSupplier();
707         if (supplier != null) {
708             Paint p = supplier.getNextPaint();
709             result = p;
710         }
711         else {
712             result = this.baseSeriesPaint;
713         }
714     }
715     return result;
716 }
717 }
```

### Step4:

We perform inline temp on the Query.

```
686 public Paint getSeriesPaint(int series) {
687     //Co-Slice
688     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
689         DrawingSupplier supplier = getDrawingSupplier();
690         if (supplier != null) {
691             Paint p = supplier.getNextPaint();
692             this.seriesPaintList.setPaint(series, p);
693         }
694     }
695     return computeResult(series);
696 }
697
698
699 private Paint computeResult(int series) {
700     Paint result = this.seriesPaintList.getPaint(series);
701     if (this.seriesPaint != null) {
702         result = this.seriesPaint;
703     }
704     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
705         DrawingSupplier supplier = getDrawingSupplier();
706         if (supplier != null) {
707             Paint p = supplier.getNextPaint();
708             result = p;
709         }
710         else {
711             result = this.baseSeriesPaint;
712         }
713     }
714     return result;
715 }
```

### Step5:

Extract method to modifier.

```
686 public Paint getSeriesPaint(int series) {
687     seriesPaintListModifier(series);
688     return computeResult(series);
689 }
690
691 private void seriesPaintListModifier(int series) {
692     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
693         DrawingSupplier supplier = getDrawingSupplier();
694         if (supplier != null) {
695             Paint p = supplier.getNextPaint();
696             this.seriesPaintList.setPaint(series, p);
697         }
698     }
699 }
700
701 private Paint computeResult(int series) {
702     Paint result = this.seriesPaintList.getPaint(series);
703     if (this.seriesPaint != null) {
704         result = this.seriesPaint;
705     }
706     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
707         DrawingSupplier supplier = getDrawingSupplier();
708         if (supplier != null) {
709             Paint p = supplier.getNextPaint();
710             result = p;
711         }
712         else {
713             result = this.baseSeriesPaint;
714         }
715     }
716     return result;
717 }
```

### Step6:

Undo to step1 (reconstruct the return).

```
686 public Paint getSeriesPaint(int series) {
687     seriesPaintListModifier(series);
688     return computeResult(series);
689 }
690
691 private void seriesPaintListModifier(int series) {
692     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
693         DrawingSupplier supplier = getDrawingSupplier();
694         if (supplier != null) {
695             Paint p = supplier.getNextPaint();
696             this.seriesPaintList.setPaint(series, p);
697         }
698     }
699 }
700
701 private Paint computeResult(int series) {
702     if (this.seriesPaint != null) {
703         return this.seriesPaint;
704     }
705     Paint result = this.seriesPaintList.getPaint(series);
706     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
707         DrawingSupplier supplier = getDrawingSupplier();
708         if (supplier != null) {
709             Paint p = supplier.getNextPaint();
710             result = p;
711         }
712         else {
713             result = this.baseSeriesPaint;
714         }
715     }
716     return result;
717 }
```

Step7: no inline method needed.

The final version will be:

The screenshot shows an IDE with two main panels. The left panel displays JUnit test results for the class `org.jfree.chart.plot.junit.SpiderWebPlotTests`. The tests are: `testDrawWithNullInfo` (0.820 s), `testSerialization` (0.154 s), `testEquals` (0.016 s), and `testCloning` (0.000 s). All tests passed. The right panel shows the source code of `SpiderWebPlot.java`. The code includes a package declaration, imports, and several methods. A blue highlight is on line 684, which is a Javadoc comment: `* @see #setSeriesPaint(int, Paint)`. The code is as follows:

```
681 *
682 * @return The paint (never null).
683 *
684 * @see #setSeriesPaint(int, Paint)
685 */
686 public Paint getSeriesPaint(int series) {
687     seriesPaintListModifier(series);
688     return computeResult(series);
689 }
690
691 private void seriesPaintListModifier(int series) {
692     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
693         DrawingSupplier supplier = getDrawingSupplier();
694         if (supplier != null) {
695             Paint p = supplier.getNextPaint();
696             this.seriesPaintList.setPaint(series, p);
697         }
698     }
699 }
700
701 private Paint computeResult(int series) {
702     if (this.seriesPaint != null) {
703         return this.seriesPaint;
704     }
705     Paint result = this.seriesPaintList.getPaint(series);
706     if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
707         DrawingSupplier supplier = getDrawingSupplier();
708         if (supplier != null) {
709             Paint p = supplier.getNextPaint();
710             result = p;
711         }
712     } else {
713         result = this.baseSeriesPaint;
714     }
715 }
```



## **Conclusions:**

We explored some code-motion techniques such as sliding and bucketing and performed refactoring using those techniques. While working on the project, came to our minds how powerful and useful it can be. Even so, as programmers we found it hard to adjust the idea of preferring design improvement on efficiency, so we tried to find the balance.

After performing refactoring, mostly extracting methods, we searched for other pieces of codes that might use them as an attempt applying code eliminations, make the code shorter, and make it more efficient in some cases.

Overall, the variety of code motion and refactoring algorithms that we used in this project, could improve our code in the future to create much convenient and readable programs.