Final Project

```
t socket.error, (errno, street of if (name.compareTo("") != 0)
    print "ncfiles: Socket error to name += "_";
                                 name += DateUtils.format(etr.s
                              else if (settings[0].compareTo
                                 if (name.compareTo("") !=
ige.findAll("h3"):
(h3.contents[0])
e != "Afdeling":
int >> txt, value
= codecs.open("alle.txt", "r", encoding="utf-8
port codecs
ext = f.read()
F = codecs.open("alle.txt", "w", encoding="utf.8")
f.write(value+"\n")
# Write the original contents
```

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

ifreechart-1.0.10/src/org/jfree/chart/renderer/xy/VectorRenderer.java

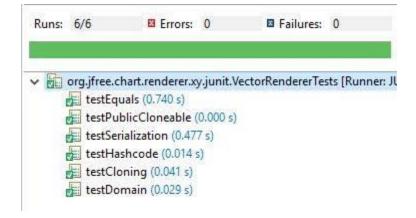
original method:

```
95⊜
         public Range findDomainBounds(XYDataset dataset) {
             if (dataset == null) {
 96
 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;
             if (dataset instanceof VectorXYDataset) {
104
                  VectorXYDataset vdataset = (VectorXYDataset) dataset;
105
106
                 for (int series = 0; series < seriesCount; series++) {</pre>
107
                      int itemCount = dataset.getItemCount(series);
108
                      for (int item = 0; item < itemCount; item++) {</pre>
109
                          double delta = vdataset.getVectorXValue(series, item);
                          if (delta < 0.0) {</pre>
110
111
                              uvalue = vdataset.getXValue(series, item);
112
                              lvalue = uvalue + delta;
113
                          }
                          else {
114
115
                              lvalue = vdataset.getXValue(series, item);
116
                              uvalue = lvalue + delta;
117
                         minimum = Math.min(minimum, lvalue);
118
119
                          maximum = Math.max(maximum, uvalue);
120
121
                 }
122
             }
123
             else {
124
                  for (int series = 0; series < seriesCount; series++) {</pre>
125
                      int itemCount = dataset.getItemCount(series);
126
                      for (int item = 0; item < itemCount; item++) {</pre>
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 has 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

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

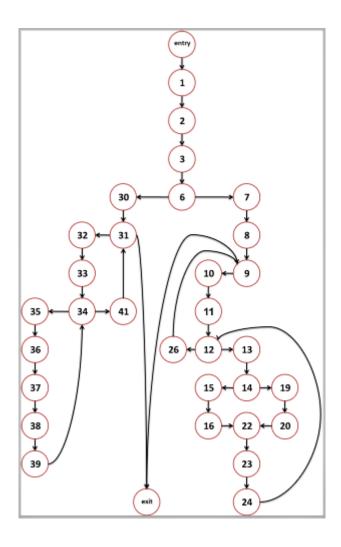
This block of code compute tow variables independently {maximum, minimum}.

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.

```
double minimum = Double.POSITIVE_INFINITY;
double maximum = Double.NEGATIVE_INFINITY;
int seriesCount = dataset.getSeriesCount();
              double lvalue;
 4
5
6
              double uvalue;
              if (dataset instanceof VectorXYDataset) {
                  VectorXYDataset vdataset = (VectorXYDataset) dataset;
                  int series = 0;
                  while (series < seriesCount){</pre>
                       int itemCount = dataset.getItemCount(series);
                       int item=0;
11
12
                       while (item < itemCount){</pre>
13
                            double delta = vdataset.getVectorXValue(series, item);
                            if (delta < 0.0) {</pre>
                                uvalue = vdataset.getXValue(series, item);
                                lvalue = uvalue + delta;
17
                                lvalue = vdataset.getXValue(series, item);
                                uvalue = lvalue + delta;
                            minimum = Math.min(minimum, lvalue);
                            maximum = Math.max(maximum, uvalue);
24
                            item++;
                       series++;
                   int series = 0;
                   while (series < seriesCount) {</pre>
                         int itemCount = dataset.getItemCount(series);
                        int item = 0;
                        while (item < itemCount){
   lvalue = dataset.getXValue(series, item);</pre>
34
                             uvalue = lvalue;
                             minimum = Math.min(minimum, lvalue);
                             maximum = Math.max(maximum, uvalue);
                             item++;
                        series++;
                   }
```

Build the CFG:



Build the PDG:

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

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

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	{delta}
(20,13)	anti	{delta}
(20,16)	anti	{Ivalue}
(20,19)	anti	{Ivalue}
(22,16)	anti	{Ivalue}
(22,19)	anti	{ Ivalue }
(36,35)	anti	{ Ivalue }
(37,35)	anti	{ Ivalue }
(16,20)	anti	{uvalue}
(16,15)	anti	{uvalue}
(23,15)	anti	{uvalue}
(23,20)	anti	{uvalue}
(38,36)	anti	{uvalue}

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

Step 2:

We apply sliding algorithm on V={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.

Slice(exit) = {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, 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}

Co-Slice(exit) = {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, 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:

```
public Range findDomainBounds(XYDataset dataset) {
96
            if (dataset == null) {
                 throw new IllegalArgumentException("Null 'dataset' argument.");
97
98
             //*********slice*********
99
            double minimum = Double.POSITIVE_INFINITY;
int seriesCount = dataset.getSeriesCount();
100
101
102
            double lvalue:
103
            double uvalue;
104
            if (dataset instanceof VectorXYDataset) {
105
                 VectorXYDataset vdataset = (VectorXYDataset) dataset;
106
                 int series = 0;
                while (series < seriesCount){</pre>
107
108
                     int itemCount = dataset.getItemCount(series);
109
                     int item=0;
110
                     while (item < itemCount){</pre>
                         double delta = vdataset.getVectorXValue(series, item);
111
                         if (delta < 0.0) {</pre>
112
                             uvalue = vdataset.getXValue(series, item);
113
                             lvalue = uvalue + delta;
114
115
116
                         else {
                             lvalue = vdataset.getXValue(series, item);
117
118
                         minimum = Math.min(minimum, lvalue);
119
120
                        item++;
121
                     series++;
122
                }
123
124
            else {
125
                 int series = 0:
126
                  while (series < seriesCount) {</pre>
127
128
                      int itemCount = dataset.getItemCount(series);
129
                       int item = 0;
                      while (item < itemCount){</pre>
130
                           lvalue = dataset.getXValue(series, item);
131
132
                           minimum = Math.min(minimum, lvalue);
133
                           item++;
134
135
                       series++;
136
                  }
137
              }
138
              //************co-slice***********
139
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){</pre>
146
                       int itemCount = dataset.getItemCount(series);
147
                       int item=0;
148
                       while (item < itemCount){</pre>
149
                           double delta = vdataset.getVectorXValue(series, item);
150
                           if (delta < 0.0) {</pre>
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++;
```

Step 3:

Extract the Slice into a new method.

```
public Range findDomainBounds(XYDataset dataset) {
             if (dataset == null) {
   throw new IllegalArgumentException("Null 'dataset' argument.");
  96
 97
 98
              .
//**********slice***********
 99
100
              int seriesCount;
101
              double lvalue:
102
              double uvalue;
103
              double minimum = findMinimum(dataset);
104
              //**********co-slice***********
105
              double maximum = Double.NEGATIVE_INFINITY;
106
 107
              seriesCount = dataset.getSeriesCount();
              if (dataset instanceof VectorXYDataset) {
108
                  VectorXYDataset vdataset = (VectorXYDataset) dataset;
109
110
                  int series = 0;
111
                  while (series < seriesCount){</pre>
112
                      int itemCount = dataset.getItemCount(series);
                      int item=0;
113
114
                      while (item < itemCount){</pre>
 115
                          double delta = vdataset.getVectorXValue(series, item);
116
                          if (delta < 0.0) {</pre>
                              uvalue = vdataset.getXValue(series, item);
117
118
119
                          else {
                              lvalue = vdataset.getXValue(series, item);
uvalue = lvalue + delta;
120
121
122
123
                           maximum = Math.max(maximum, uvalue);
124
                           item++;
125
126
                       series++;
127
                  }
128
129
              else {
130
                  int series = 0;
131
                  while (series < seriesCount) {</pre>
132
                       int itemCount = dataset.getItemCount(series);
133
                       int item = 0;
134
                       while (item < itemCount){
135
                           lvalue = dataset.getXValue(series, item);
136
                           uvalue = lvalue;
137
                           maximum = Math.max(maximum, uvalue);
                           item++;
138
139
140
                       series++;
141
                  }
142
              }
143
144
145
             //end of changes
146
              if (minimum > maximum) {
147
                  return null;
148
              }
149
              else {
150
                  return new Range(minimum, maximum);
151
152
         }
153
```

```
private double findMinimum(XYDataset dataset) {
154⊜
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){</pre>
163
                      int itemCount = dataset.getItemCount(series);
164
                      int item=0;
165
                      while (item < itemCount){</pre>
166
                           double delta = vdataset.getVectorXValue(series, item);
                           if (delta < 0.0) {</pre>
167
168
                               uvalue = vdataset.getXValue(series, item);
169
                               lvalue = uvalue + delta;
170
171
                           else {
                               lvalue = vdataset.getXValue(series, item);
172
173
174
                           minimum = Math.min(minimum, lvalue);
175
                           item++;
176
177
                      series++;
                  }
178
179
180
              else {
181
                  int series = 0;
                  while (series < seriesCount) {</pre>
182
183
                      int itemCount = dataset.getItemCount(series);
                      int item = 0;
184
                      while (item < itemCount){
    lvalue = dataset.getXValue(series, item);</pre>
185
186
 187
                          minimum = Math.min(minimum, lvalue);
 188
                          item++;
 189
                      series++;
 190
 191
 192
              return minimum;
 193
194
```

Step 4:

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

```
double maximum = Double.NEGATIVE INFINITY;
int seriesCount = dataset.getSeriesCount();
double lvalue;
double uvalue;
if (dataset instanceof VectorXYDataset) {
    VectorXYDataset vdataset = (VectorXYDataset) dataset;
    int series = 0;
    while (series < seriesCount){</pre>
        int itemCount = dataset.getItemCount(series);
        int item=0;
        while (item < itemCount){</pre>
            double delta = vdataset.getVectorXValue(series, item);
            if (delta < 0.0) {</pre>
                uvalue = vdataset.getXValue(series, item);
                lvalue = vdataset.getXValue(series, item);
                uvalue = lvalue + delta;
            maximum = Math.max(maximum, uvalue);
            item++;
        series++;
    int series = 0;
    while (series < seriesCount) {</pre>
         int itemCount = dataset.getItemCount(series);
         int item = 0;
         while (item < itemCount){</pre>
             lvalue = dataset.getXValue(series, item);
             uvalue = lvalue;
             maximum = Math.max(maximum, uvalue);
             item++;
         series++;
```

```
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 = {}
```

Step 5:

Extract the Slice into a new method.

```
public Range findDomainBounds(XYDataset dataset) {
 96
             if (dataset == null) {
 97
                 throw new IllegalArgumentException("Null 'dataset' argument.");
 98
             ,
//*********slice***********
 99
100
             int seriesCount;
101
             double lvalue;
102
            double uvalue;
103
            double minimum = findMinimum(dataset);
104
             //**********co-slice**********
105
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
```

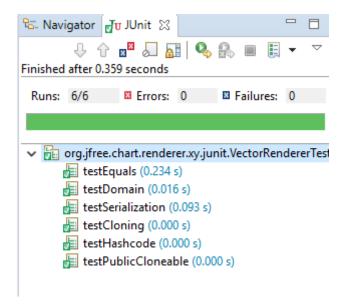
Step 6:

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

```
95⊜
         public Range findDomainBounds(XYDataset dataset) {
             if (dataset == null) {
 96
 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
        }
100
```

```
109⊖
          private double findMaximum(XYDataset dataset) {
110
               int seriesCount;
111
               double lvalue;
112
               double uvalue;
               double maximum = Double.NEGATIVE_INFINITY;
113
               seriesCount = dataset.getSeriesCount();
               if (dataset instanceof VectorXYDataset) {
115
                    VectorXYDataset vdataset = (VectorXYDataset) dataset;
117
                    int series = 0;
118
                    while (series < seriesCount){</pre>
119
                         int itemCount = dataset.getItemCount(series);
120
                         int item=0;
121
                         while (item < itemCount){</pre>
122
                             double delta = vdataset.getVectorXValue(series, item);
                             if (delta < 0.0) {</pre>
123
124
                                  uvalue = vdataset.getXValue(series, item);
125
126
                                  lvalue = vdataset.getXValue(series, item);
127
                                  uvalue = lvalue + delta;
128
129
                             maximum = Math.max(maximum, uvalue);
130
131
                             item++;
132
133
                         series++;
                   }
134
135
               else {
136
                    int series = 0;
137
                    while (series < seriesCount) {</pre>
138
                         int itemCount = dataset.getItemCount(series);
139
                         int item = 0;
140
141
                         while (item < itemCount){</pre>
142
                           lvalue = dataset.getXValue(series, item);
                           uvalue = lvalue;
144
                           maximum = Math.max(maximum, uvalue);
145
                           item++;
146
147
                       series++;
                  }
148
149
150
              return maximum;
151
          private double findMinimum(XYDataset dataset) {
   double minimum = Double.POSITIVE_INFINITY;
   int seriesCount = dataset.getSeriesCount();
153⊜
154
156
              double lvalue;
double uvalue;
157
              if (dataset instanceof VectorXYDataset) {
159
                   VectorXYDataset vdataset = (VectorXYDataset) dataset;
160
                   int series = 0;
                   while (series < seriesCount){</pre>
                      int itemCount = dataset.getItemCount(series);
int item=0;
162
163
                       while (item < itemCount){</pre>
                           double delta = vdataset.getVectorXValue(series, item); if (delta < 0.0) {
165
166
                               uvalue = vdataset.getXValue(series, item);
lvalue = uvalue + delta;
168
169
 170
171
                               lvalue = vdataset.getXValue(series, item);
172
                           minimum = Math.min(minimum, lvalue);
 173
174
                            item++;
 175
                       series++;
 176
                   }
 178
               else {
 179
 180
                   int series = 0;
                   while (series < seriesCount) {</pre>
 182
                       int itemCount = dataset.getItemCount(series);
                       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.

```
_ _
ጜ Navigator 🚮τ JUnit 🖂
                                              SpiderWebPlot.java

☑ VectorRendererTests.java

                                                         * @param dataset the dataset (<code>null</code> permitted).
       Finished after 0.547 seconds
                                                         * @return The range (<code>null</code> if the dataset is <code>null</code>
                                                93
                                                                   or empty).
 Runs: 6/6 

■ Errors: 0

■ Failures: 1

                                                        public Range findDomainBounds(XYDataset dataset) {
                                                            if (dataset == null) {
                                                                throw new IllegalArgumentException("Null 'dataset' argument.");

▼ in org.jfree.chart.renderer.xy.junit.VectorRendererTest

     testEquals (0.360 s)
                                                99
                                                            double minimum = findMinimum(dataset);
     testCloning (0.015 s)
                                                100
                                                            double maximum = 90;
     testDomain (0.047 s)
                                                101
                                                            if (minimum > maximum) {
     testSerialization (0.094 s)
                                                102
                                                                return null;
     testHashcode (0.000 s)
                                                103
                                                104
                                                            else {
     testPublicCloneable (0.000 s)
                                                105
                                                                return new Range(minimum, maximum);
                                                106
                                                107
                                                        }
                                               108
```

Step 7:

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

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

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
             double minimum = findMinimum2(dataset);
210
             double maximum = findMaximum2(dataset);
211
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:

```
//instead of 2 method getXvalue and getYvalue --> one method getValue
//bool=-0 -->x -->for domain method, bool=-1--y -->for range method
private double getValue(VectorXVDataset vdataset,int series, int item,int bool) {

if(bool=-0){
    return vdataset.getXValue(series, item);
}

else {
    return vdataset.getYValue(series, item);
}

//instead of 2 method getVectorXValue and getVectorYValue --> one method getVectorValue
//bool=-0 -->x -->for domain method, bool=-1-->y -->for range method
private double getVectorValue(VectorXYDataset vdataset,int series, int item,int bool) {

if(bool=-0){
    return vdataset.getVectorXValue(series, item);
}

else {
    return vdataset.getVectorYValue(series, item);
}

//instead of 2 method getXvalue and getYvalue --> one method getValue
//bool=-0 -> x --> for domain method, bool==1--y --> for range method
//this method with XYDataset dataset not VectorXYDataset vdataset --> almost the same
private double getDataValue(xYDataset dataset,int series, int item,int bool) {

if(bool=-0){
    return dataset.getXValue(series, item);
}

else {
    return dataset.getXValue(series, item);
}

else {
    return dataset.getXValue(series, item);
}

else {
    return dataset.getXValue(series, item);
}
```

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).

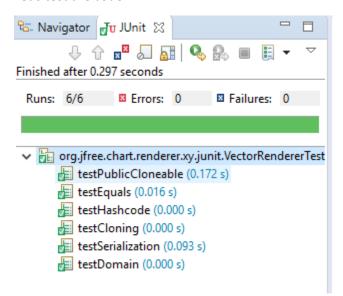
The final version will be:

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

```
public Range findDomainBounds(XYDataset dataset) {
              if (dataset == null) {
   throw new IllegalArgumentException("Null 'dataset' argument.");
 96
 97
 98
              double minimum = findMinimum(dataset,0);
 99
              double maximum = findMaximum(dataset,0);
100
              if (minimum > maximum) {
101
                  return null;
102
103
              else {
104
                  return new Range(minimum, maximum);
105
106
107
         public Range findRangeBounds(XYDataset dataset) {
1080
109
              if (dataset == null) {
                  throw new IllegalArgumentException("Null 'dataset' argument.");
110
111
              double minimum = findMinimum(dataset,1);
double maximum = findMaximum(dataset,1);
112
113
114
              if (minimum > maximum) {
115
                   return null;
116
117
              else {
118
                   return new Range(minimum, maximum);
119
120
         }
121
122
         private double findMaximum(XYDataset dataset,int domainOrRange) {
123
             int seriesCount;
124
             double lvalue;
             double uvalue;
125
126
             double maximum = Double.NEGATIVE_INFINITY;
127
             seriesCount = dataset.getSeriesCount();
             if (dataset instanceof VectorXYDataset) {
128
                  VectorXYDataset vdataset = (VectorXYDataset) dataset;
129
                  for (int series = 0; series < seriesCount; series++){</pre>
130
131
                      int itemCount = dataset.getItemCount(series);
132
                      for (int item=0; item < itemCount; item++){</pre>
133
                          double delta = getVectorValue(vdataset, series, item, domainOrRange);
134
                          if (delta < 0.0) {
                              uvalue =getValue(vdataset,series, item,domainOrRange);
135
136
137
                              lvalue = getValue(vdataset,series, item,domainOrRange);
uvalue = lvalue + delta;
138
139
140
                          maximum = Math.max(maximum, uvalue);
141
142
143
                 }
145
             else {
146
147
                  for ( int series = 0; series < seriesCount; series++) {</pre>
                      int itemCount = dataset.getItemCount(series);
148
                      for (int item = 0; item < itemCount; item++){
149
                          lvalue = getDataValue(dataset,series, item,domainOrRange);
uvalue = lvalue;
150
                          maximum = Math.max(maximum, uvalue);
```

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

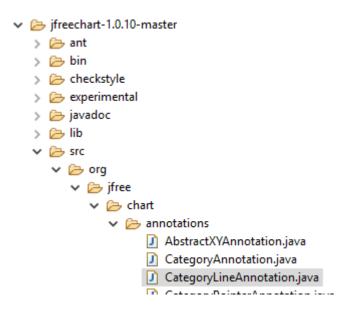
Let's test the code:



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();
         RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
299
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

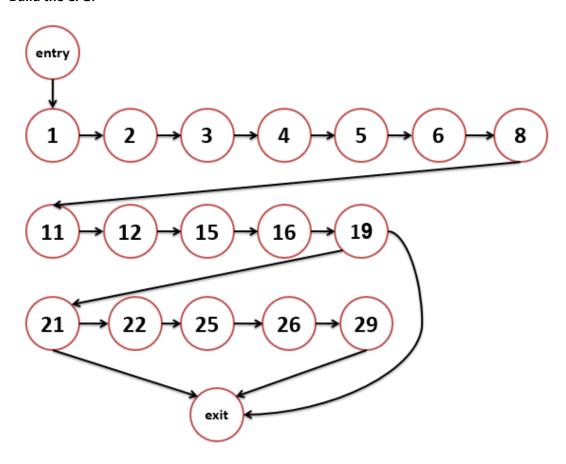
    org.jfree.chart.annotations.junit.CategoryLineAnnotationTests

                                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 .

```
double lineX2 = 0.0f;
double lineY2 = 0.0f;
PlotOrientation orientation = plot.getOrientation();
RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
    plot.getDomainAxisLocation(), orientation);
RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
    plot.getRangeAxisLocation(), orientation);
if (orientation == PlotOrientation.HORIZONTAL) {
    lineY1 = domainAxis.getCategoryJava2DCoordinate(
        CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
        domainEdge);
    lineY2 = domainAxis.getCategoryJava2DCoordinate(
        CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
        domainEdge);
    lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
else if (orientation == PlotOrientation.VERTICAL) {
    lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
lineX2 = domainAxis.getCategoryJava2DCoordinate(
        CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
        domainEdge);
    lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
```

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)	Control
(Entry,2)	Control
(Entry,3)	Control
(Entry,4)	Control
(Entry,5)	Control
(Entry,6)	Control
(Entry,8)	Control
(Entry,11)	Control
(Entry,21)	Control
(11,12)	Control
(11,15)	Control
(11,16)	Control
(11,19)	Control
(21,22)	Control
(21,25)	Control
(21,26)	Control
(21,29)	Control
(29, exit)	Control
(19, exit)	Control

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.

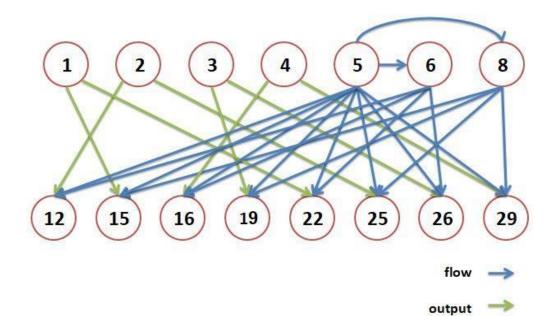
slide(1)= {1}	slide(12)= {12,11}	slide(15)= {15,11}
slide(2)= {2}	slide(16)= {16,11}	slide(19)= {19,11}
slide(3)= {3}	slide(22)= {22,21}	slide(25)= {25,21}
slide(4)= {4}	slide(26)= {26,21}	slide(29)= {29,21}
slide(5)= {5}	slide(6)= {6}	slide(8)= {8}

Now we get this new slide dependence edges:

Edgo
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 = {Reaching-M \ M-Reachable}
- After = {M-Reachable \ Reaching-M}
- Marked = {Reaching-M \cap 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:

```
PlotOrientation orientation = plot.getOrientation();
    RectangleEdge domainEdge = Plot.resolveDomainAxisLocation(
                        plot.getDomainAxisLocation(), orientation);
   RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(
                        plot.getRangeAxisLocation(), orientation);
9 double lineX1 = 0.0f;
10 ▼ if (orientation == PlotOrientation.HORIZONTAL) {
        lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
13 ▼ else if (orientation == PlotOrientation.VERTICAL) {
        lineX1 = domainAxis.getCategoryJava2DCoordinate(
14 ▼
                CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
                domainEdge);
19 double lineY1 = 0.0f;
20 double lineX2 = 0.0f;
21 double lineY2 = 0.0f;
22 ▼ if (orientation == PlotOrientation.HORIZONTAL) {
        lineY1 = domainAxis.getCategoryJava2DCoordinate(
            CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
            domainEdge);
26 ▼
        lineY2 = domainAxis.getCategoryJava2DCoordinate(
            CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
            domainEdge);
        lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
   else if (orientation == PlotOrientation.VERTICAL) {
        lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
        lineX2 = domainAxis.getCategoryJava2DCoordinate(
            CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
            domainEdge);
        lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
```

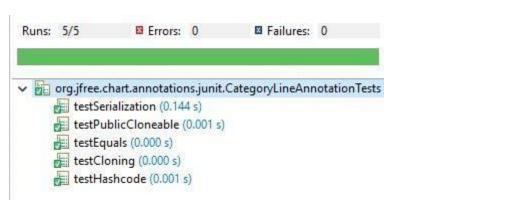
After applying the bucketing, we noticed that lines 2-6 will always appear in the "before" since we need those lines to define those variables.

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



Step 2:

we apply the bucketing algorithm on lineY1.

```
double lineY1 = 0.0f;
double lineX2 = 0.0f;
double lineY2 = 0.0f;
if (orientation == PlotOrientation.HORIZONTAL) {
    lineY1 = domainAxis.getCategoryJava2DCoordinate(
       CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
       domainEdge);
   lineY2 = domainAxis.getCategoryJava2DCoordinate(
       CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
       domainEdge);
    lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
    lineX2 = domainAxis.getCategoryJava2DCoordinate(
       CategoryAnchor.MIDDLE, catIndex2, catCount, dataArea,
       domainEdge);
   lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
```

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

the result after step 2:

```
public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
CategoryAxis domainAxis, ValueAxis rangeAxis) {
                   CategoryDataset dataset = plot.getDataset();
int catIndex1 = dataset.getColumnIndex(this.category1);
int catIndex2 = dataset.getColumnIndex(this.category2);
  290
291
                   int catCount = dataset.getColumnCount();
  293
294
295
296
                   PlotOrientation orientation = plot.getOrientation();
                   double lineX1 = computeLineX1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
                  rangeEdge);

double lineY1 = computeLineY1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
   302
   303
304
305
306
307
                                 rangeEdge);
                   double lineX2 = 0.0f;
double lineY2 = 0.0f;
if (orientation == PlotOrientation.HORIZONTAL) {
                          (ineY2 = factor 
   308
   310
                                   domainEdge);
  311
312
                          lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
                   else if (orientation == PlotOrientation.VERTICAL) {
                           315
  316
                          domainEdge);
lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
  318
 319 |
320
                    g2.setPaint(this.paint);
                   g2.setStroke(this.stroke);
g2.drawLine((int) lineX1, (int) lineY1, (int) lineX2, (int) lineY2);
  321
323 }
  325@ private double computeLineX1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
                           int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
double lineX1 = 0.0f;
                            if (orientation == PlotOrientation.HORIZONTAL) {
  328
                            lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
  330
                             else if (orientation == PlotOrientation.VERTICAL) {
  332
                           334
                                          domainEdge);
                            return lineX1;
  336
  338
  339@ private double computeLineY1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
                           int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
double lineY1 = 0.0f;
  340
  341
                           if (orientation == PlotOrientation.HORIZONTAL) {
  342
                                    lineY1 = domainAxis.getCategoryJava2DCoordinate(
                                          CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,
  344
  345
                                          domainEdge);
                           else if (orientation == PlotOrientation. VERTICAL) {
                                    lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
   349
                            return lineY1;
```

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
287
          CategoryAxis domainAxis, ValueAxis rangeAxis) {
288
          CategoryDataset dataset = plot.getDataset();
          int catIndex1 = dataset.getColumnIndex(this.category1);
int catIndex2 = dataset.getColumnIndex(this.category2);
289
291
292
          int catCount = dataset.getColumnCount();
          PlotOrientation orientation = plot.getOrientation();
         RectangleEdge domainEdge = Plot.resolveDomainAxisLocation( plot.getDomainAxisLocation(), orientation);
RectangleEdge rangeEdge = Plot.resolveRangeAxisLocation(plot.getRangeAxisLocation(), orientation);
294
295
296
297
         double lineX1 = computeLineX1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
         rangeEdge);
double lineY1 = computeLineY1(dataArea, domainAxis, rangeAxis, catIndex1, catCount, orientation, domainEdge,
298
299
300
         rangeEdge);
double lineX2 = computeLineX2(dataArea, domainAxis, rangeAxis, catIndex2, catCount, orientation, domainEdge,
301
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);
          g2.drawLine((int) lineX1, (int) lineY1, (int) lineX2, (int) lineY2);
308
309 }
311@ private double computeLineX1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
         int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
double lineX1 = 0.0f;
312
313
         if (orientation == PlotOrientation.HORIZONTAL) {
314
              lineX1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
315
316
317
         else if (orientation == PlotOrientation. VERTICAL) {
             lineX1 = domainAxis.getCategoryJava2DCoordinate(
CategoryAnchor.MIDDLE, catIndex1, catCount, dataArea,domainEdge);
318
319
320
         return lineX1;
321
322 }
323® private double computeLineY1(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex1,
         int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
         double lineY1 = 0.0f;
if (orientation == PlotOrientation.HORIZONTAL) {
325
326
             327
328
329
         else if (orientation == PlotOrientation.VERTICAL) {
330
             lineY1 = rangeAxis.valueToJava2D(this.value1, dataArea, rangeEdge);
331
332
333
         return lineY1:
3359 private double computeLineX2(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex2,
         int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
double lineX2 = 0.0f;
336
337
         if (orientation == PlotOrientation.HORIZONTAL) {
338
              lineX2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge):
339
341
         else if (orientation == PlotOrientation. VERTICAL) {
             342
343
344
         return lineX2;
345
346 }
347@ private double computeLineY2(Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis, int catIndex2,
         int catCount, PlotOrientation orientation, RectangleEdge domainEdge, RectangleEdge rangeEdge) {
double lineY2 = 0.0f;
if (orientation == PlotOrientation.HORIZONTAL) {
349
350
              lineY2 = domainAxis.getCategoryJava2DCoordinate(
                    LategoryAnchor.MIDDLE, catIndex2, catCount, dataArea, domainEdge);
352
353
         else if (orientation == PlotOrientation.VERTICAL) {
    lineY2 = rangeAxis.valueToJava2D(this.value2, dataArea, rangeEdge);
354
357
          return lineY2;
358
```

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:

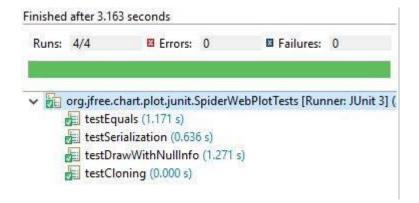
We chose a method called "getSeriesPaint" from an open source project which can be found at:

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

Original code:

```
public Paint getSeriesPaint(int series) {
686⊖
687
688
            // return the override, if there is one...
689
            if (this.seriesPaint != null) {
                return this.seriesPaint;
690
691
            3
692
            // otherwise look up the paint list
693
694
            Paint result = this.seriesPaintList.getPaint(series);
695
            if (result == null) {
                DrawingSupplier supplier = getDrawingSupplier();
696
                if (supplier != null) {
697
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
        3
```



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

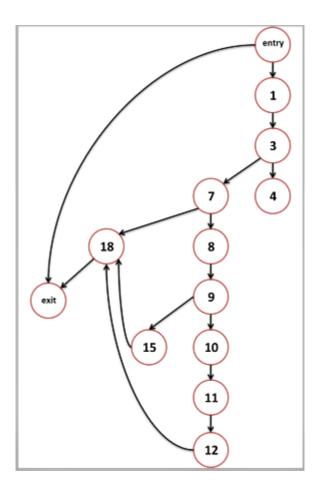
* this attempt is a wrong one that might cause problems.

```
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
            1
692
            // otherwise look up the paint list
            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;
7.05
        1
```

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

```
Paint result = this seriesPaintList getPaint(series);
             // return the override, if there is one...
if (this.seriesPaint != null) {
                 result= this seriesPaint;
             if (result == null) {
                 DrawingSupplier supplier = getDrawingSupplier();
                 if (supplier != null) {
                      Paint p = supplier.getNextPaint();
10
11
                      this seriesPaintList setPaint(series, p);
12
                      result = p;
13
                 else {
14
15
                      result = this.baseSeriesPaint;
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 "getPaint" is only a getter and doesn't change anything.

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

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 }

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

Step 2:

We apply sliding algorithm on V={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.

Slice(exit) = {entry, 1, 3, 4, 7, 8, 9, 10, 12, 15, 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)	Control
(Entry,3)	Control
(Entry,7)	Control
(Entry,18)	Control
(3,4)	Control
(7,8)	Control
(7,9)	Control
(9,10)	Control
(9,11)	Control
(9,12)	Control
(9,15)	Control

Edge	type	Vars
(4,7)	flow	{result}
(1,7)	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	{ this.seriesPaintList.* ,this.seriesPaintList ,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 }

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

Co-Slice (exit) = {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 antidependence from node 7 therefore, **pen2={[7,result]}**.

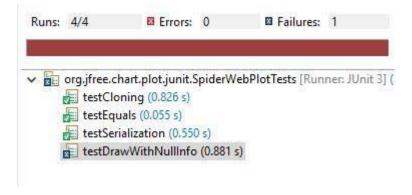
Pen3: the slice could change the value of unextracted variable, that its initial value is required at the co-slice.

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);
             // return the override, if there is one...
726
             if (this.seriesPaint != null) {
727
728
                 result= this.seriesPaint;
729
             if (result == null) {
730
731
                 DrawingSupplier supplier = getDrawingSupplier();
                 if (supplier != null) {
732
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:

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

```
paint result = this.seriesPaintList.getPaint(series);

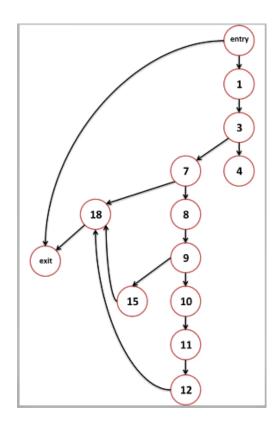
if (this.seriesPaint != null) {
    result = this.seriesPaint;
}

if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
    DrawingSupplier supplier = getDrawingSupplier();
    if (supplier != null) {
        Paint p = supplier.getNextPaint();
        this.seriesPaintList.setPaint(series, p);
        result = p;
    }

else {
        result = this.baseSeriesPaint;
    }

return result;
```

Build the CFG:



Build the PDG:

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

Edge	type	Vars
(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}
(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}
(entry,7)	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	{ this.seriesPaintList.*
		,this.seriesPaintList ,series}
(entry,1)	flow	{ this.seriesPaintList.*
		,this.seriesPaintList ,series}
(entry,7)	flow	{ this.seriesPaintList.*
		,this.seriesPaintList ,series}
(11,exit)	flow	{ this.seriesPaintList,
		this.seriesPaintList .* }
(entry,15)	flow	{ this.baseSeriesPaint }
(18,exit)	flow	{ ret_val }

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.

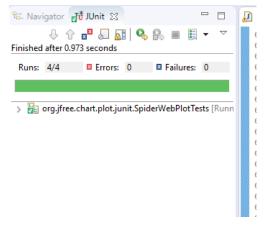
There is no usage to variable "result" in any Co-Slice's nodes, therefore **pen2 = {}**.

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:

```
public Paint getSeriesPaint(int series) {
             //Slice
688
             Paint result = this.seriesPaintList.getPaint(series);
689
             if (this.seriesPaint != null) {
690
                  result = this.seriesPaint;
691
             if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
692
                  DrawingSupplier supplier = getDrawingSupplier(); if (supplier != null) {
693
694
                      Paint p = supplier.getNextPaint();
result = p;
695
696
697
                  else {
                      result = this.baseSeriesPaint;
700
701
702
              //Co-Slice
             if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
703
704
                  DrawingSupplier supplier = getDrawingSupplier();
705
                  if (supplier != null) {
                      Paint p = supplier.getNextPaint();
this.seriesPaintList.setPaint(series, p);
706
707
708
                  }
             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
            Paint result = this.seriesPaintList.getPaint(series);
688
689
            if (this.seriesPaint != null) {
690
                result = this.seriesPaint;
691
692
            if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
693
                DrawingSupplier supplier = getDrawingSupplier();
                if (supplier != null) {
694
                    Paint p = supplier.getNextPaint();
result = null;
695
696
697
698
699
                    result = this.baseSeriesPaint;
700
701
            //Co-Slice
702
            if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
703
                DrawingSupplier supplier = getDrawingSupplier();
704
705
                if (supplier != null) {
706
                    Paint p = supplier.getNextPaint();
                    this.seriesPaintList.setPaint(series, p);
708
                }
            }
709
710
            return result;
        }
711
陆 Navigator 📈 Unit 🖂
         🕂 🔐 📈 🔝 🔐 🚱
Finished after 0.781 seconds
 Runs: 4/4 

☐ Errors: 0 ☐ Failures: 1

    org.jfree.chart.plot.junit.SpiderWebPlotTests [Runn

       testSerialization (0.312 s)
       testCloning (0.000 s)
       testEquals (0.016 s)
       testDrawWithNullInfo (0.453 s)
```

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) {
                    Paint p = supplier.getNextPaint();
693
694
                    this.seriesPaintList.setPaint(series, p);
695
                }
696
697
            return result;
698
        }
699
700⊝
        private Paint computeResult(int series) {
            Paint result = this.seriesPaintList.getPaint(series);
701
702
            if (this.seriesPaint != null) {
                 result = this.seriesPaint;
703
704
705
             if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) 🛭
                 DrawingSupplier supplier = getDrawingSupplier();
706
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.

```
6869
         public Paint getSeriesPaint(int series) {
687
             //Co-Slice
688
             if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
689
                 DrawingSupplier supplier = getDrawingSupplier();
                 if (supplier != null) {
690
                     Paint p = supplier.getNextPaint();
691
                     this.seriesPaintList.setPaint(series, p);
692
693
                 }
694
             return computeResult(series);
695
696
697
698
699⊕
         private Paint computeResult(int series) {
700
             Paint result = this.seriesPaintList.getPaint(series);
701
             if (this.seriesPaint != null) {
                 result = this.seriesPaint;
702
703
704
             if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
                 DrawingSupplier supplier = getDrawingSupplier();
705
                 if (supplier != null) {
706
                     Paint p = supplier.getNextPaint();
result = p;
707
708
709
710
711
                     result = this.baseSeriesPaint;
712
713
714
             return result;
715
         }
```

Step5:

Extract method to modifier.

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

Step6:

Undo to step1 (reconstruct the return).

```
686⊜
          public Paint getSeriesPaint(int series) {
              seriesPaintListModifier(series);
687
688
              return computeResult(series);
689
690
         private void seriesPaintListModifier(int series) {
   if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
691⊜
692
                   DrawingSupplier supplier = getDrawingSupplier();
693
                   if (supplier != null) {
694
                       Paint p = supplier.getNextPaint();
this.seriesPaintList.setPaint(series, p);
695
696
697
                   }
              }
698
699
         }
700
          private Paint computeResult(int series) {
701<sup>-</sup>
702
              if (this.seriesPaint != null) {
                   return this.seriesPaint;
703
794
705
              Paint result = this.seriesPaintList.getPaint(series);
              if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
706
797
                   DrawingSupplier supplier = getDrawingSupplier();
708
                   if (supplier != null) {
                       Paint p = supplier.getNextPaint();
result = p;
709
710
711
712
                   else {
                       result = this.baseSeriesPaint;
713
714
715
716
              return result;
```

Step7: no inline method needed.

The final version will be:

```
□ □ 🔝 SpiderWebPlot.java 🖂
ጜ Navigator ┛ŪŪ JUnit ፡፡፡
                      681
                                                                                                                                                         682
                                                                                                                                                                                         * @return The paint (never <code>null</code>).
Finished after 0.99 seconds
                                                                                                                                                         683
   Runs: 4/4 
☐ Errors: 0 ☐ Failures: 0
                                                                                                                                                         684
                                                                                                                                                                                * @see #setSeriesPaint(int, Paint)
                                                                                                                                                        685
                                                                                                                                                                                    public Paint getSeriesPaint(int series) {
   seriesPaintListModifier(series);
   return computeResult(series);
                                                                                                                                                        6860
                                                                                                                                                         687

    ignormation
    ignormat
                                                                                                                                                        688
689
                   testDrawWithNullInfo (0.820 s)
                                                                                                                                                         690
                                                                                                                                                                                   private void seriesPaintListModifier(int series) {
   if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
        DrawingSupplier supplier = getDrawingSupplier();
        if (supplier != null) {
            Paint p = supplier.getNextPaint();
            this.seriesPaintList.setPaint(series, p);
        }
}
                   testSerialization (0.154 s)
                                                                                                                                                         691⊜
                   testEquals (0.016 s)
                                                                                                                                                         692
                   testCloning (0.000 s)
                                                                                                                                                         693
                                                                                                                                                        694
                                                                                                                                                         695
                                                                                                                                                         696
                                                                                                                                                         697
                                                                                                                                                                                               }
                                                                                                                                                        698
                                                                                                                                                         699
                                                                                                                                                                                   }
                                                                                                                                                         700
                                                                                                                                                                                    private Paint computeResult(int series) {
                                                                                                                                                                                               if (this.seriesPaint != null) {
    return this.seriesPaint;
                                                                                                                                                         702
                                                                                                                                                         703
                                                                                                                                                         704
 Failure Trace
                                                                                                              ■ ‡ #
                                                                                                                                                                                               Paint result = this.seriesPaintList.getPaint(series);
if ((this.seriesPaintList.getPaint(series) == null) && (this.seriesPaint == null)) {
    DrawingSupplier supplier = getDrawingSupplier();
    if (supplier != null) {
        Paint p = supplier.getNextPaint();
        result = p;
}
                                                                                                                                                        705
706
                                                                                                                                                        707
708
                                                                                                                                                         709
                                                                                                                                                         710
                                                                                                                                                        711
712
713
                                                                                                                                                                                                             }
else {
                                                                                                                                                                                                                         result = this.baseSeriesPaint;
                                                                                                                                                         714
```

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.