Solution
Assignment 06 — 21.10.2020 – v1.0
Software Visualization

Please submit this exercise by email to pascal.gadient@inf.unibe.ch before 28 October 2020, 10:15am.

You must submit your code as editable text, i.e., use plain text file(s).

For this exercise, we need Pharo 9.0 (instead of the previous 8.0 releases). If not already done, download the latest stand-alone release of *Pharo Launcher* from here and install the application. After it is successfully installed, start the application and click on "New" (top left). In the new window that appears, choose "Official distributions" and "Pharo 9.0 - 64bit (development version, latest)" (or the corresponding 32bit release if your CPU or OS does not support 64 bits). Click on "Create image". Select the newly created *Pharo* entry from the list and click on "Launch". A new window that runs Pharo will be displayed.

Next, we have to enable additional feature support in Roassal3, e.g., for the method `numberOfLinesOfCode`. For that, in the main screen of Pharo 9.0 click on the menu "Tools", then "Roassal3", and finally on "Load full version". This process can take several minutes depending on your device's CPU and internet connection. We advise you to save the image when the installation succeeded to avoid redoing this process.

Your task is to create plots that look as similar as possible to those presented in each exercise, including the colors and spacings.

Troubleshooting:

1. Problem (macOS only):
   Launcher won't launch.

   Solution:
   Change the launcher setting to launch with login shell (query "login" in the settings and uncheck "Launch image from a login shell")

2. Problem (macOS only):
   Launcher won't launch.

   Solution:
   Acknowledge the dialog where the app asks for trust.

Exercise 1: Sunburst visualization with Roassal (2 pts)

Build a Sunburst visualization as shown in Figure 1 to analyze the test coverage of the `Collection` class hierarchy. Each tile represents a specific class, and the size of the tile should represent its number of lines of code. Moreover, tested classes (i.e., classes covered by tests) should be colored in green, while other classes should remain in grey.

Hint: You can assume that test classes (i.e., classes that test other classes) use a name which closely resembles the original name of the class they test; in general, they add only the postfix `Test` to the original class name (e.g., `ByteArray` will become to `ByteArrayTest`).

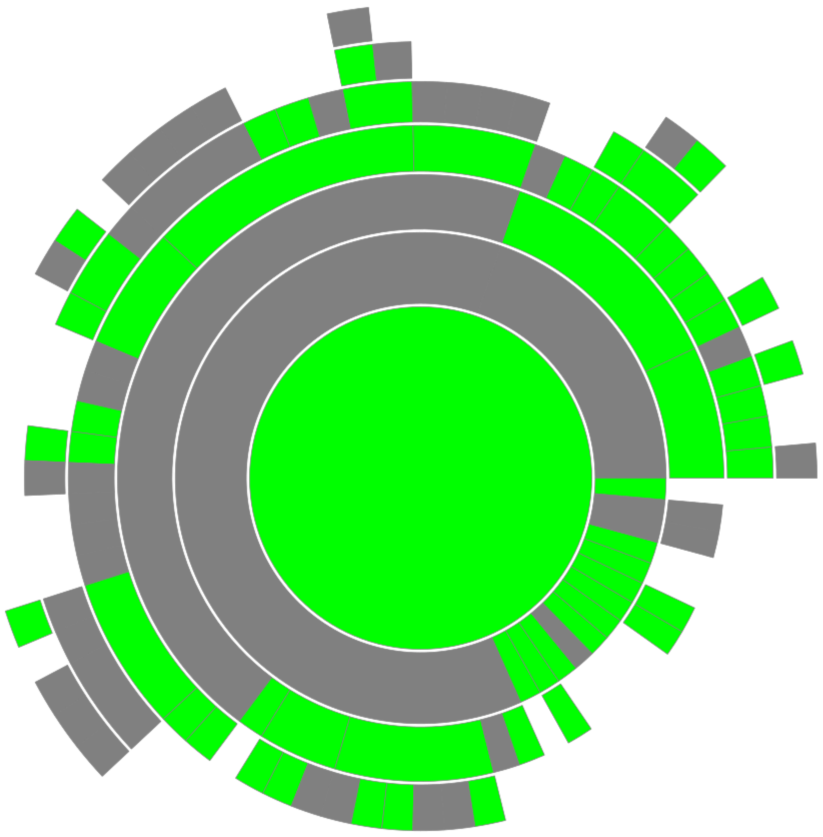

Figure 1: Sunburst visualization built with Roassal

Answer:

```
| sb |
sb := RSSunburstBuilder new.
sb sliceShape withBorder.
sb sliceColor: [:shape |
   (Smalltalk includesKey: (shape model name , 'Test') asSymbol)
   ifTrue: [ Color green ]
   ifFalse: [ Color gray ] ].
sb explore: Collection using: #subclasses.
RSNormalizer size
   shapes: (sb shapes select: #isSLeaf);
   normalize: #numberOfLinesOfCode.
sb build.
sb canvas @ RSCanvasController.
^ sb canvas
```

Exercise 2: Tree layout visualization with Roassal (2 pts)

Build a tree as shown in Figure 2 to highlight subclasses of the class `Collection`, which have again subclasses and contain the string `Array` in their names. Circles should be used to represent the classes, and the size of each circle should encode the number of methods of the represented class. Moreover, classes that have subclasses and contain the string `Array` in their names must be colored green, whereas other classes must remain grey.

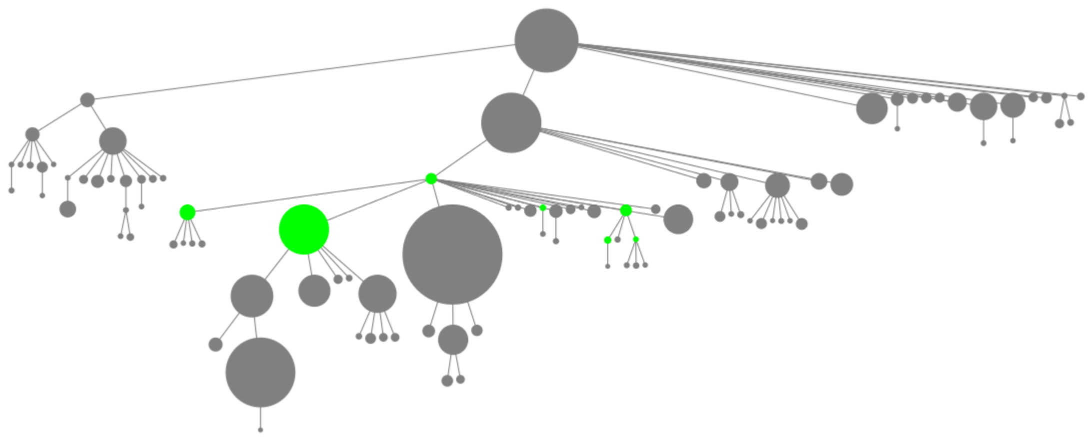

Figure 2: Tree layout visualization built with Roassal

Answer:

```
| canvas shapes |
canvas := RSCanvas new.
shapes := (Collection withAllSubclasses ) collect: [ :class |
  (class subclasses notEmpty and: ['*Array*' match: class name])
  ifTrue: [RSEllipse new model: class; color: Color green; popup ]
  ifFalse: [RSEllipse new model: class; color: Color gray; popup ]].
canvas addAll: shapes.
RSEdgeBuilder line
  canvas: canvas;
  connectFrom: [ :class | class superclass ].
RSNormalizer size
  shapes: shapes;
  normalize: [ :cls | cls numberOfMethods].
RSTreeLayout on: shapes.
canvas edges pushBack.
canvas zoomToFit.
^ canvas
```

Exercise 3: Node-link visualization with Roassal (3 pts)

In this exercise you have to create a node-link visualization as shown in Figure 3 to analyze the class dependencies between the `Collection` class hierarchy and the `RSLayout` class hierarchy. To this end, you have to:

i) Visualize the classes of both hierarchies using circles (i.e., `RSEllipse`)

ii) Use the red (`Collection`) and green (`RSLayout`) color to highlight the classes of each hierarchy.

iii) Add edges to depict the class hierarchy, while using the `RSClusterLayout`

iv) Add blue Bézier edges to depict class dependencies using `RSMultiBezierEdgeBuilder`

v) Map the number of methods of each class to its circle size using `RSNormalizer`

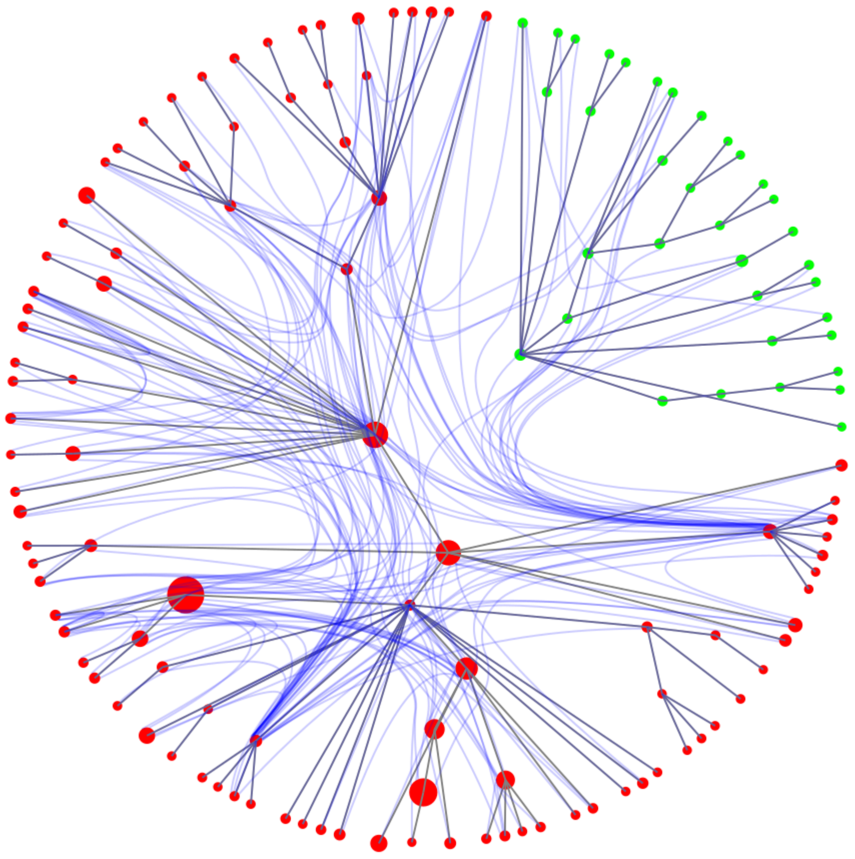
Figure 3: Node-link visualization built with Roassal

Answer:

```
| c classes |
c := RSCanvas new.
classes := RSLayout withAllSubclasses , Collection withAllSubclasses.
classes := classes asSet
  collect: [ :cls | (RSLayout withAllSubclasses includes: cls)
    ifTrue: [RSEllipse new model: cls; color: Color green]
    ifFalse: [RSEllipse new model: cls; color: Color red]]
  as: RSGroup.
c addAll: classes.
RSEdgeBuilder line
  color: Color gray;
  canvas: c;
  shapes: classes;
  connectFrom: #superclass.
RSNormalizer size
  shapes: classes;
  to: 20;
  normalize: #numberOfMethods.
RSClusterLayout on: classes.
RSMultiBezierEdgeBuilder multiBezier
  borderColor: (Color blue alpha: 0.2);
  canvas: c;
  shapes: classes;
  withBorderAttachPoint;
  following: #superclass;
  connectToAll: #dependentClasses.
c @ RSCanvasController.
classes @ RSPopup.
^ c
```

Exercise 4: Discussion (3 pts)

Comment on the strenghts and limitations of each visualization you just created.
Answer:

Sunburst visualization. *The sunburst visualization provides a nice representation for hierarchies; each ring from the center moving outwards corresponds to a level in the hierarchy. The rings are sliced into numerous tiles, each of them revealing the relationship to the parent tile. Besides those advantages, anything related to the covered area of specific tiles is hard to evaluate manually, since humans cannot accurately estimate areas of circular sections.*

Tree layout visualization. *The tree layout visualization uses more space and is not as compact as the Sunburst visualization. This effect requires more navigation. However, it is easier to analyze the classes, relationships, and classes at the various levels in the hierarchy.*

Node-link visualization. *Node-link visualizations combine the best of both worlds: they support multiple overlaid relation visualizations (in our example using Bézier curves and straight lines) and represent the hierarchy level of each element in an intuitive manner. Unfortunately, these plots require a very high resolution to be of any use, because of their heavy utilization of tiny connections between the different shapes.*