Working with sets

Table of contents

1. On the use of sets in Mascopt

1.1. How things work

If you use mascopt, there's a chance that you will need to deal with sets. Mascopt has sets implemented in a rather convenient way with some features that make programmer life easier but, if misunderstood, can lead to a debugging nightmare. This part of the documentation tries to explain how to correctly use the sets for everyday programming.

At first the use of sets is pretty intuitive. You can construct them, put some elements into them (e.g. vertices or edges as mascopt tries to be a graph library), remove some elements, traverse the set, etc... Things get a little bit trickier as you may want to create subsets of already existing sets. In order to keep a certain number of thing coherents, when you create a subset *S'* of an already existing set *S*, automatically *S'* will "observe" *S*. That is, if you remove an element from *S*, it will be automatically removed from *S'* and you will only be able to add in *S'* elements that already exist in *S*. This feature is very convenient but makes things a little bit more difficult to debug (from my experience).

1.2. Programming with sets

In order to illustrate the previous paragraph, let's play a little bit with vertex sets. We are going to write a program that constructs a vertex set *V*, a sub-vertex set of *V*, *V'* and another set *V"* contaning the same edges than *V*. And finally we will remove and add edges to see what happens.

1.3. Creating the sets

First we create the set *V0* and put some vertices in it and display the content of the set,

```
VertexSet V0 = new VertexSet();

for (int i=0; i<10;i++)
{
  V0.add(new Vertex(Math.random(),Math.random()));
}
System.out.println("V0: "+V0);
```

The output is be something like the following:

```
V0: { N8 , N5 , N9 , N3 , N1 , N6 , N7 , N0 , N2 , N4 }
```

The we create a subset *V1* of *V0* and display its content,

```
VertexSet V1 = new VertexSet(V0);
System.out.println("V1: "+V1);
```

The output will then be something of the following form:

```
V1: { N8 , N5 , N9 , N3 , N1 , N6 , N7 , N0 , N2 , N4 }
```

As you can see, it is not very different than previously. Note that all vertices are directly added in the subset. And now we create a vertex set *V2* independant from the previous 2 sets but with the same vertices in it.

```
VertexSet V2 = new VertexSet(V0,true);
System.out.println("V2: "+V2);
```

The display is something that looks like:

```
V2: { N8 , N5 , N9 , N3 , N1 , N6 , N7 , N0 , N2 , N4 }
```

The order of the vertices may not be the same.

1.4. Removing some elements

Now that the different sets are created, we are going to delete and add some elements in them. First we pick an element of *V0* and remove it :

```
itV0 = V0.iterator();
V0.remove((Vertex)itV0.next());
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

The output is then:

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V1: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N6 , N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

The vertex *N6* has been removed in both sets *V0* and *V1* thanks to the observation mechanism. *V1* is a subset of *V0*, /i.e./ *V1* observes *V0*. If this time we remove an element from *V1*, only *V1* will be affected:

```
Iterator itV1 = V1.iterator();
V1.remove((Vertex)itV1.next());
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

The output is, this time,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N6 , N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

And finally, if we remove an edge from *V2*, as it is independant from the other sets, it will be the only one affected:

```
Iterator itV2 = V2.iterator();
V2.remove((Vertex)itV2.next());
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

And naturally, as expected, the output is,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

1.5. Adding some elements

This time we are creating some vertices and we want to add them in our already existing sets. This operation is really easy to perform for the sets *V0* and *V2*, first we create the vertices,

```
Vertex newVertex1 = new Vertex(Math.random(),Math.random());
Vertex newVertex2 = new Vertex(Math.random(),Math.random());
```

and then we add them

```
V0.add(newVertex1);
V2.add(newVertex2);
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

and we get as an output,

```
V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N10 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
V2: { N7 , N11 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

But, as *V1* is a subset of *V0* and mascopt guaranties that it will remain so, you can't add anything, only elements already present in *V0*,

```
V1.add(newVertex1);
V1.add(newVertex2);
System.out.println();
System.out.println("V0: "+V0);
System.out.println("V1: "+V1);
System.out.println("V2: "+V2);
```

gives the following output,

```

V0: { N7 , N8 , N5 , N2 , N1 , N9 , N4 , N10 , N0 , N3 }
V1: { N8 , N5 , N2 , N1 , N9 , N4 , N10 , N0 , N3 }
V2: { N7 , N11 , N8 , N5 , N2 , N1 , N9 , N4 , N0 , N3 }
```

1.6. Last remarks

- In order to create the set *V2* we could have created a new set and have placed the vertices in it ourselves but the result would have been exactly the same.
- You may have remarked that in order to traverse the sets we use an object called *Iterator*. This is made possible thanks to the internal use of an *HashSet* to store the elements and it is particularly interesting for two main reasons,
  - the loop is very easy to write
  - if an element appears or disappears from the set, an old *Iterator* is no more usable and automatically points to *null*, thus indicating a change

2. Sample

The above program can be found in MascoptDev: samples/basics/SetsUse.java. To test it, just do:

```

java SetsUse
```