CarRecall specification examaple

This example shows how Mango exposes cases splits to the user, if desired. Once generated, the specification may be navigated by opening folders in the Mango Explorer and following links in the revealed specification views.

Start by loading the FirstYearCode project.

On MacOSX, go to Eclipse>Preferences>"Mango Preferences". For other operating systems: Window>Preferences>"Mango Preferences".

Hit "Restore Defaults", in particular, "Suppress advice dialogs" and "Do not pause for case-splits" should be unchecked.

Hit OK to close the preferences.

In the Package Explorer, in the FirstYearCode project, open src/firstYearCode/CarRecall.java.

In the CarRecall editor, double-click on "CarRecall" in the class declaration line "public class CarRecall {". This will select both CarRecall and the CarRecall editor tab.

Right-click, Mango>populate. When the Mango button becomes green, right-click, Mango>run.

At the first case split, Mango will notify the user of a pause. (see figure)

```
32 public class CarRecall {
33     public static void main(String[] args){
34         int modelNum;
35         Scanner input = new Scanner (System.in);
36         do{
37             System.out.println("Enter the car's model number or 0 to quit: ");
38             modelNum=input.nextInt();
39
40             //test if car is defective
41             if ((modelNum==119)||(modelNum==179)
42                     ||((modelNum>=189)&&(modelNum<=195))
43                     ||(modelNum==221)||(modelNum==780))
44             {
45                 System.out.println("Your car is defective. It must be repaired.");
46             }
47             else if (modelNum==0)
```

Case split advice

Mango has paused. You may view the posted case splits. Press the Mango button when ready to proceed. (You can change the Mango preferences to defeat this advice or run Mango without pausing.)

OK

Console | Specification | Problems | Search | &1 | &2

modelNum does not equal 780

input assumptions

```
modelNum does not equal 780
modelNum does not equal 0
outinputStaticArea is defined
```

output state

heap

stack

Click OK, the Mango Explorer box will turn green. After a while it is easier just to when the Mango button changes color and not bother with the notification. Check "Suppress advice dialogs" to defeat notification in the future, if desired.

The line of source code corresponding to the case split has been highlighted. Observe the two cases, named &1 and &2, by clicking on their tabs. Each specification view is a functional description, and as such has two parts, "input assumptions" and "output state". For the "modelNum equals 780" case, the heap may be opened to reveal that "Your car is defective" has been appended to the "out" buffer. (see figure).

```
40              //test if car is defective
41              if ((modelNum==119)||(modelNum==179)
42                      ||((modelNum>=189)&&(modelNum<=195))
43                      ||(modelNum==221)||(modelNum==780))
44              {
45                  System.out.println("Your car is defective. It must be repaired.");
46              }
47              else if (modelNum==0)
48              {
49                  System.out.println ("Program terminated.");
50              }
51              else
52              {
53                  System.out.println ("Your car is not defective.");
54              }
55          }while (modelNum!=0);
56      }
57  }
58
```

Problems | Specification | Console | &1 | &2

modelNum equals 780

input assumptions

```
modelNum equals 780
outinputStaticArea is defined
```

output state

heap

<localVar> at unresolved location

```
outinputStaticArea.buffer=outinputStaticArea.buffer + Your car is defective. It must be
repaired.<nl>
```

stack

Likewise, in the case "modelNum does not equal 780", the heap contains "Your car is not defective". So far so good, but what happened to the case modelNum==0, yielding "Program terminated". The short answer is that Mango hasn't forgotten about it, but isn't ready to consider it either. More specifically, Mango currently is building the specification for the loop body. The case modeNum==0 is a loop termination condition which does not yield a state transition for the loop body, so it is not relevant at the moment.

Spend some time assimilating the correspondence between procedural programming, the java code, and functional programming, the specification views. When you are ready, hit the Mango button so that Mango can proceed to the next case (see figure).

```
            modelNum=input.nextInt();

            //test if car is defective
            if ((modelNum==119)||(modelNum==179)
                    ||((modelNum>=189)&&(modelNum<=195))
                    ||(modelNum==221)||(modelNum==780))
            {
                System.out.println("Your car is defective. It must be repaired.");
            }
            else if (modelNum==0)
            {
                System.out.println ("Program terminated.");
            }
            else
            {
                System.out.println ("Your car is not defective.");
            }
        }while (modelNum!=0);
    }
}
```

Problems | Specification | Console | firstYearCode.CarRecall.main([Ljava/lang/String;)V.loop | &1 | &2

modelNum does not equal 221

input assumptions

modelNum does not equal 221
Does modelNum equal 780?

output state

heap

<localVar> at unresolved location

outinputStaticArea.buffer=outinputStaticArea.buffer + var1

stack

Observe that if modelNum does not equal 221, then execution drops through to the case Mango has already analyzed. This is because when we read code, we proceed top down in execution order. But the Mango functional analysis is always bottom up in the reverse of execution order. Put more simply, Mango starts at the end and works backwards through the case-splits. Notice the input assumptions "Does modelNum equal 780?". The technical interpretation of this assumption goes like this: "This case satisfies the input assumptions for the nested case "Does modelNum equal 780". The reason for this shorthand notation is pragmatic. To explicitly say what those assumptions are quickly leads to exponential growth in the size of the specification. Trust me on this, I have been there. Fortunately, there is a simpler, more useful interpretation: "The output state for this case will have variables in it referencing the nested case "Does modelNum equal 780?".

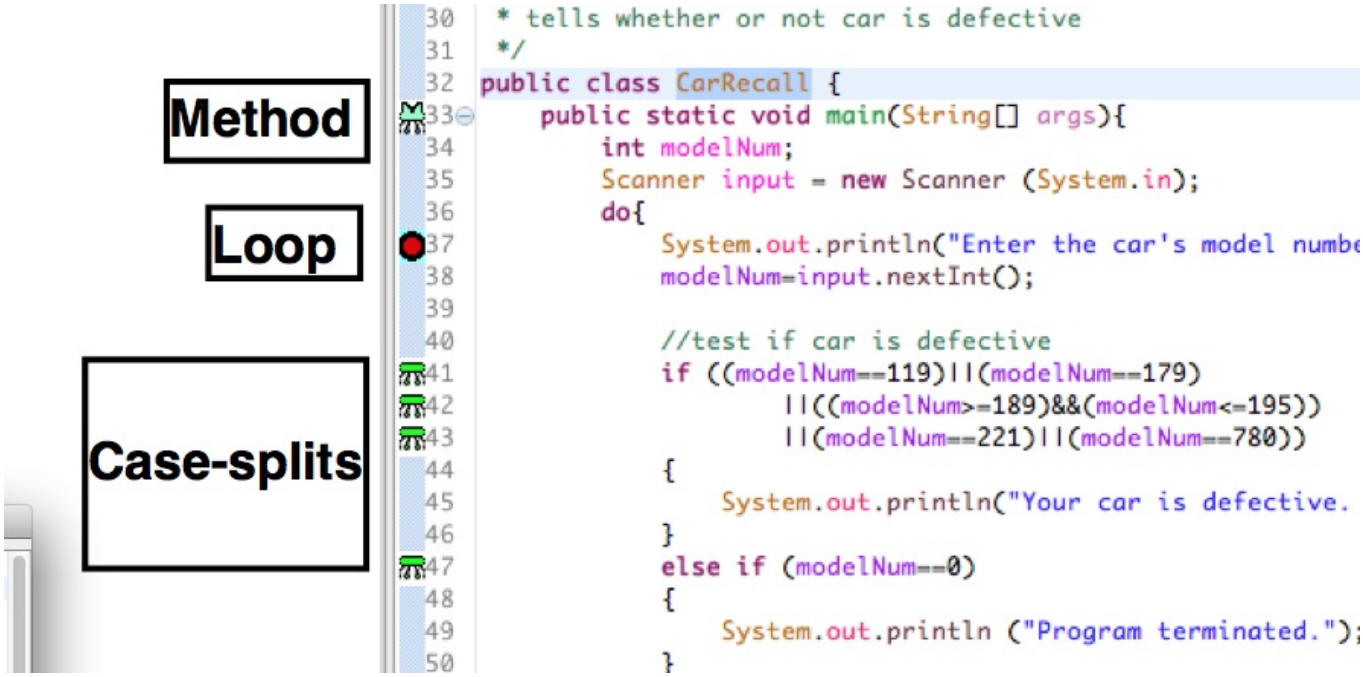
Indeed, observe that the heap has such a variable, "var1". Go ahead and click on it now (see figure).

Problems | Specification | Console | firstYearCode.CarRecall.main([Ljava/lang/String;)V.loop | &1 | &2

modelNum does not equal 221

```
var1
Re: Does modelNum equal 780?

yes: Your car is defective. It must be repaired.<nl>
no : Your car is not defective.<nl>
```

A new panel appears for this view which give a reprise of the heap values for the nested case, tied to the possible input assumptions for that case. In general, if there is any doubt as the the validity of the values a variable may achieve in any given situation, it is always possible to open it up and see what is going on. If you find yourself doing this for more than two or three levels, then its possible that the code is not very well thought out, or perhaps you don't have a good grip on the nested cases in the first place.

The Mango analysis will continue enumerating case splits, but for now let's fast-forward to the end of the specification. In the Mango preferences panel, check "Suppress advice dialogs" and "Do not pause". Now press the Mango button, causing Mango to run without interruption. You can monitor the progress of the run by viewing the console window. During the run, icons for the various case-splits will appear. Near the end of the run, a "loop icon" will appear in the source code icon gutter, and then finally the method icon (see figure).