

## Assignment 1 - Internal DSL

Source code: [github.com/simonmdsn/mdsd/tree/main/assignment-1](https://github.com/simonmdsn/mdsd/tree/main/assignment-1)

Classes: [github.com/simonmdsn/mdsd/tree/main/assignment-1/bin](https://github.com/simonmdsn/mdsd/tree/main/assignment-1/bin)

Tests: [github.com/simonmdsn/mdsd/tree/main/assignment-1/test](https://github.com/simonmdsn/mdsd/tree/main/assignment-1/test)

### Solving the assignment

I chose the advanced assignment and created a similar DSL to [gitlab.sdu.dk/mdsd](https://gitlab.sdu.dk/mdsd) in my favorite programming language Dart. Dart is hybrid of Java and JavaScript making it easy to read for anyone familiar with either language.

I tried to be as true to the Java skeleton to make it easier to make use of the test cases. My assignment passes all tests from the skeleton, albeit I translated all the test cases from Java to Dart. The translation was seamless with some IDE magic.

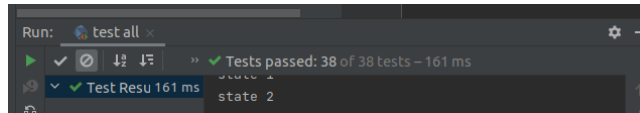


Figure 1: Passing all tests with command `dart test`.

I solved the exercise with a reverse engineering approach from the test cases and classes. As an example the code in listing 1 from `CDPlayerTest.java`, reveals that a latest state is needed to be tracked in the `StateMachine` class to actually set the stop state as the initial state for this particular instance. Furthermore, the state playing needs to be created by the `to("PLAYING")` method call, even though `state("PLAYING")` is called later in the method chain. Therefore, both the `to` and `state` methods need to take into account if the states already exists or not. This was the iterative process for most of the methods.

```
1 stateMachine.  
2     integer("track").  
3     state("STOP").initial()  
4         when("PLAY").to("PLAYING").set("track", 1)  
5             .ifEquals("track", 0).  
6         when("PLAY").to("PLAYING").  
7     state("PLAYING")
```

Listing 1: Test code from `CDPlayerTest.java`.

## Source code

```
1 extension FirstWhereOrNullExtension<E> on Iterable<E> {
2     E? firstWhereOrNull(bool Function(E) test) {
3         for (E element in this) {
4             if (test(element)) return element;
5         }
6         return null;
7     }
8 }
```

```
1
2 import 'state.dart';
3
4 class Machine {
5     final List<State> states = [];
6     final Map<String, int> integers = {};
7     late State initialState;
8
9     Machine();
10 }
```

```
1 import 'transition.dart';
2
3 class State {
4     final String name;
5
6     final List<Transition> transitions = [];
7
8     State({required this.name});
9 }
```

```
1 import 'state.dart';
2
3 class Transition {
4     final String event;
5     late State? targetState;
6     late Operation? operation;
7     late int? operationValue;
8     late String? operationVariableName;
9     late Condition? condition;
10    late String? conditionalVariableName;
11    late int? conditionValue;
12
13    Transition({
14        required this.event,
15        this.targetState,
16        this.operation,
17        this.operationValue,
18        this.operationVariableName,
19        this.condition,
20        this.conditionVariableName,
21        this.conditionValue,
22    });
23 }
24
```

```
25 enum Operation {
26   set,
27   increment,
28   decrement,
29 }
30
31 enum Condition {
32   equal,
33   greater,
34   less,
35 }
36
37 import 'extensions.dart';
38 import 'machine.dart';
39 import 'state.dart';
40 import 'transition.dart';
41
42 class MachineInterpreter {
43   late State currentState;
44   late Machine machine;
45
46   MachineInterpreter({required this.machine}) {
47     currentState = machine.initialState;
48     for (var element in machine.states) {
49       print(element.name);
50     }
51   }
52
53   factory MachineInterpreter.run(Machine machine) {
54     return MachineInterpreter(machine: machine);
55   }
56
57   void processEvent(String event) {
58     var where =
59       currentState.transitions.where((element) => element.event
60 == event);
61     final transition = where.firstWhereOrNull((element) {
62       var conditionsSatisfied = true;
63       if (element.condition != null &&
64         element.conditionalVariableName != null &&
65         element.conditionValue != null) {
66         switch (element.condition) {
67           case Condition.equal:
68             conditionsSatisfied = element.conditionValue! ==
69               machine.integers[element.conditionalVariableName!];
69           break;
70           case Condition.greater:
71             conditionsSatisfied =
72               machine.integers[element.conditionalVariableName!]!
73 >
74               element.conditionValue!;
75           break;
76           case Condition.less:
77             conditionsSatisfied =
78               machine.integers[element.conditionalVariableName!]!
79 <
80               element.conditionValue!;
81           break;
82         }
83       }
84     });
85     if (transition != null) {
86       currentState = transition.targetState;
87     }
88   }
89 }
```

```
44         default:
45             throw Exception(
46                 'Something went wrong when validating transition
conditions');
47         }
48     }
49     if (element.operation != null && conditionsSatisfied) {
50         switch (element.operation) {
51             case Operation.increment:
52                 machine.integers[element.operationVariableName!] =
53                     machine.integers[element.operationVariableName!] +
54                     1;
55                 break;
56             case Operation.decrement:
57                 machine.integers[element.operationVariableName!] =
58                     machine.integers[element.operationVariableName!] -
59                     1;
60                 print(machine.integers[element.operationVariableName!]);
61                 break;
62             case Operation.set:
63                 machine.integers[element.operationVariableName!] =
64                     element.operationValue!;
65                 break;
66             default:
67                 throw Exception(
68                     'Something went wrong when validating transition
operations');
69             }
70         }
71         return conditionsSatisfied;
72     });
73     if (transition != null) {
74         currentState = transition.targetState!;
75     }
76 }
77
78 int getInteger(String string) {
79     return machine.integers[string]!;
80 }
81 }
```

```
1 import 'extensions.dart';
2 import 'machine.dart';
3 import 'state.dart';
4 import 'transition.dart';
5
6 class StateMachine {
7     final Machine machine = Machine();
8     State? latestState;
9     Transition? latestTransition;
10
11     Machine build() {
12         return machine;
13     }
14
15     StateMachine state(String state) {
16         var stateWhere = machine.states.firstWhereOrNull((p0) => p0.
name == state);
```

```
17     if (stateWhere == null) {
18         stateWhere = State(name: state);
19         machine.states.add(stateWhere);
20     }
21     latestState = stateWhere;
22     return this;
23 }
24
25 StateMachine initial() {
26     machine.initialState = machine.states.last;
27     return this;
28 }
29
30 StateMachine when(String state) {
31     var transition = Transition(event: state, targetState: null);
32     latestState!.transitions.add(transition);
33     latestTransition = transition;
34     return this;
35 }
36
37 StateMachine to(String state) {
38     var stateWhere = machine.states.firstWhereOrNull((p0) => p0.
39         name == state);
40     if (stateWhere == null) {
41         stateWhere = State(name: state);
42         machine.states.add(stateWhere);
43     }
44     latestTransition!.targetState = stateWhere;
45     return this;
46 }
47
48 StateMachine integer(String string) {
49     machine.integers[string] = 0;
50     return this;
51 }
52
53 StateMachine set(String variableName, int integer) {
54     latestTransition!.operation = Operation.set;
55     latestTransition!.operationValue = integer;
56     latestTransition!.operationVariableName = variableName;
57     return this;
58 }
59
60 StateMachine increment(String variableName) {
61     latestTransition!.operation = Operation.increment;
62     latestTransition!.operationVariableName = variableName;
63     return this;
64 }
65
66 StateMachine decrement(String variableName) {
67     latestTransition!.operation = Operation.decrement;
68     latestTransition!.operationVariableName = variableName;
69     return this;
70 }
71
72 StateMachine ifEquals(String name, int comparison) {
73     latestTransition!.conditionalVariableName = name;
```

```
73     latestTransition!.conditionValue = comparison;
74     latestTransition!.condition = Condition.equal;
75     return this;
76 }
77
78 StateMachine ifGreaterThan(String name, int comparison) {
79     latestTransition!.conditionalVariableName = name;
80     latestTransition!.conditionValue = comparison;
81     latestTransition!.condition = Condition.greater;
82     return this;
83 }
84
85 StateMachine ifLessThan(String name, int comparison) {
86     latestTransition!.conditionalVariableName = name;
87     latestTransition!.conditionValue = comparison;
88     latestTransition!.condition = Condition.less;
89     return this;
90 }
91 }
```