Assignment 1 - Internal DSL

Source code: github.com/simonmdsn/mdsd/tree/main/assignment-1
Classes: github.com/simonmdsn/mdsd/tree/main/assignment-1/bin
Tests: 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 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.

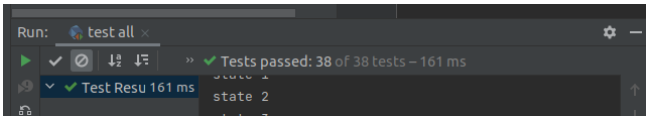

Figure 1: Passing all tests with command `dart test`.

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

```dart
extension FirstWhereOrNullExtension<E> on Iterable<E> {
  E? firstWhereOrNull(bool Function(E) test) {
    for (E element in this) {
      if (test(element)) return element;
    }
    return null;
  }
}
```

```dart

import 'state.dart';

class Machine {
  final List<State> states = [];
  final Map<String, int> integers = {};
  late State initialState;

  Machine();
}
```

```dart
import 'transition.dart';

class State {
  final String name;

  final List<Transition> transitions = [];

  State({required this.name});
}
```

```dart
import 'state.dart';

class Transition {
  final String event;
  late State? targetState;
  late Operation? operation;
  late int? operationValue;
  late String? operationVariableName;
  late Condition? condition;
  late String? conditionalVariableName;
  late int? conditionValue;

  Transition({
    required this.event,
    this.targetState,
    this.operation,
    this.operationValue,
    this.operationVariableName,
    this.condition,
    this.conditionalVariableName,
    this.conditionValue,
  });
}

```

```
enum Operation {
  set,
  increment,
  decrement,
}

enum Condition {
  equal,
  greater,
  less,
}
```

```
import 'extensions.dart';
import 'machine.dart';
import 'state.dart';
import 'transition.dart';

class MachineInterpreter {
  late State currentState;
  late Machine machine;

  MachineInterpreter({required this.machine}) {
    currentState = machine.initialState;
    for (var element in machine.states) {
      print(element.name);
    }
  }

  factory MachineInterpreter.run(Machine machine) {
    return MachineInterpreter(machine: machine);
  }

  void processEvent(String event) {
    var where =
        currentState.transitions.where((element) => element.event
    == event);
    final transition = where.firstWhereOrNull((element) {
      var conditionsSatisfied = true;
      if (element.condition != null &&
          element.conditionalVariableName != null &&
          element.conditionValue != null) {
        switch (element.condition) {
          case Condition.equal:
            conditionsSatisfied = element.conditionValue! ==
                machine.integers[element.conditionalVariableName!];
            break;
          case Condition.greater:
            conditionsSatisfied =
                machine.integers[element.conditionalVariableName!]!
    >
                    element.conditionValue!;
            break;
          case Condition.less:
            conditionsSatisfied =
                machine.integers[element.conditionalVariableName!]!
    <
                    element.conditionValue!;
            break;
```

```
          default:
            throw Exception(
                'Something went wrong when validating transition conditions');
          }
        }
        if (element.operation != null && conditionsSatisfied) {
          switch (element.operation) {
            case Operation.increment:
              machine.integers[element.operationVariableName!] =
                  machine.integers[element.operationVariableName]! + 1;
              break;
            case Operation.decrement:
              machine.integers[element.operationVariableName!] =
                  machine.integers[element.operationVariableName]! - 1;
              print(machine.integers[element.operationVariableName]);
              break;
            case Operation.set:
              machine.integers[element.operationVariableName!] =
                  element.operationValue!;
              break;
            default:
              throw Exception(
                  'Something went wrong when validating transition operations');
          }
        }
        return conditionsSatisfied;
      });
      if (transition != null) {
        currentState = transition.targetState!;
      }
    }

    int getInteger(String string) {
      return machine.integers[string]!;
    }
}
```

```
import 'extensions.dart';
import 'machine.dart';
import 'state.dart';
import 'transition.dart';

class StateMachine {
  final Machine machine = Machine();
  State? latestState;
  Transition? latestTransition;

  Machine build() {
    return machine;
  }

  StateMachine state(String state) {
    var stateWhere = machine.states.firstWhereOrNull((p0) => p0.name == state);
```

```
    if (stateWhere == null) {
      stateWhere = State(name: state);
      machine.states.add(stateWhere);
    }
    latestState = stateWhere;
    return this;
  }

  StateMachine initial() {
    machine.initialState = machine.states.last;
    return this;
  }

  StateMachine when(String state) {
    var transition = Transition(event: state, targetState: null);
    latestState!.transitions.add(transition);
    latestTransition = transition;
    return this;
  }

  StateMachine to(String state) {
    var stateWhere = machine.states.firstWhereOrNull((p0) => p0.
    name == state);
    if (stateWhere == null) {
      stateWhere = State(name: state);
      machine.states.add(stateWhere);
    }
    latestTransition!.targetState = stateWhere;
    return this;
  }

  StateMachine integer(String string) {
    machine.integers[string] = 0;
    return this;
  }

  StateMachine set(String variableName, int integer) {
    latestTransition!.operation = Operation.set;
    latestTransition!.operationValue = integer;
    latestTransition!.operationVariableName = variableName;
    return this;
  }

  StateMachine increment(String variableName) {
    latestTransition!.operation = Operation.increment;
    latestTransition!.operationVariableName = variableName;
    return this;
  }

  StateMachine decrement(String variableName) {
    latestTransition!.operation = Operation.decrement;
    latestTransition!.operationVariableName = variableName;
    return this;
  }

  StateMachine ifEquals(String name, int comparison) {
    latestTransition!.conditionalVariableName = name;
```

```
    latestTransition!.conditionValue = comparison;
    latestTransition!.condition = Condition.equal;
    return this;
  }

  StateMachine ifGreaterThan(String name, int comparison) {
    latestTransition!.conditionalVariableName = name;
    latestTransition!.conditionValue = comparison;
    latestTransition!.condition = Condition.greater;
    return this;
  }

  StateMachine ifLessThan(String name, int comparison) {
    latestTransition!.conditionalVariableName = name;
    latestTransition!.conditionValue = comparison;
    latestTransition!.condition = Condition.less;
    return this;
  }
}
```