Assignment 1: internal DSL

By Peter Brændgaard, pebra18

# Summary

I solved the assignment by first implementing all the getters with appropriate attributes. I then looked at the remaining methods and implemented simple logic, so that the program would compile. I then finished the assignment through an iterative process, where I ran a test and made changes to the program until it succeeded. This was repeated until all tests passed.

# No. of tests passed

All

# Repository link

<https://github.com/Peterzxcvbnm/Model-Driven-development/tree/main/assignment1-main>

# Source code

package main.metamodel;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
public class Machine {  
  
 private List<State> states = new ArrayList<>();  
 private State initialState;  
 private HashMap<String, Integer> integers = new HashMap<>();  
  
 public List<State> getStates() {  
 return states;  
 }  
  
 public void setInitialState(State initialState) {  
 this.initialState = initialState;  
 }  
  
 public State getInitialState() {  
 return initialState;  
 }  
  
 public State getState(String name) {  
 return states.stream().filter(x -> x.getName().equals(name)).findAny().orElse(null);  
 }  
  
 public void addInteger(String name){  
 integers.put(name, 0);  
 }  
  
 public int numberOfIntegers() {  
 return integers.size();  
 }  
  
 public boolean hasInteger(String name) {  
 return integers.containsKey(name);  
 }  
  
 public HashMap<String, Integer> getIntegers(){  
 return integers;  
 }  
  
  
}

package main.metamodel;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class State {  
  
 private final String name;  
 private final Machine machine;  
 private final ArrayList<Transition> transitions;  
  
 public State(String name, Machine machine) {  
 this.name = name;  
 this.machine = machine;  
 this.transitions = new ArrayList<>();  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public List<Transition> getTransitions() {  
 return transitions;  
 }  
  
 public Transition getTransitionByEvent(String string) {  
 return transitions  
 .stream()  
 .filter(x -> x.getEvent().equals(string))  
 .filter(x -> !x.isConditional() ||  
 x.mayHappen(machine.getIntegers().get(x.getConditionVariableName())))  
 .findAny()  
 .orElse(null);  
  
 }  
  
  
}

package main.metamodel;  
  
public class Transition {  
  
 private String eventName;  
 private String targetName;  
 private State target;  
 private OperationType operationType = OperationType.*None*;  
 private String operationVariableName;  
 private int operationValue;  
 private String conditionalVariableName;  
 private int conditionComparedValue;  
 private ConditionType conditionType;  
  
 public Transition(String eventName) {  
 this.eventName = eventName;  
 }  
  
 public String getEvent() {  
 return eventName;  
 }  
  
 public void setTargetName(String target) {  
 this.targetName = target;  
 }  
  
 public String getTargetName() {  
 return targetName;  
 }  
  
 public void setTarget(State target) {  
 this.target = target;  
 }  
  
 public State getTarget() {  
 return target;  
 }  
  
 public void setOperationType(OperationType operationType) {  
 this.operationType = operationType;  
 }  
  
 public boolean hasSetOperation() {  
 return operationType.equals(OperationType.*Set*);  
 }  
  
 public boolean hasIncrementOperation() {  
 return operationType.equals(OperationType.*Increment*);  
 }  
  
 public boolean hasDecrementOperation() {  
 return operationType.equals(OperationType.*Decrement*);  
 }  
  
 public void setOperationVariableName(String operationVariableName) {  
 this.operationVariableName = operationVariableName;  
 }  
  
 public String getOperationVariableName() {  
 return operationVariableName;  
 }  
  
 public int getOperationValue() {  
 return operationValue;  
 }  
  
 public void setOperationValue(int operationValue) {  
 this.operationValue = operationValue;  
 }  
  
 public void setConditionalVariableName(String conditionalVariableName) {  
 this.conditionalVariableName = conditionalVariableName;  
 }  
  
 public void setConditionComparedValue(int conditionComparedValue) {  
 this.conditionComparedValue = conditionComparedValue;  
 }  
  
 public void setConditionType(ConditionType conditionType) {  
 this.conditionType = conditionType;  
 }  
  
 public boolean isConditional() {  
 return conditionalVariableName != null;  
 }  
  
 public String getConditionVariableName() {  
 return conditionalVariableName;  
 }  
  
 public int getConditionComparedValue() {  
 return conditionComparedValue;  
 }  
  
 public boolean isConditionEqual() {  
 return conditionType == ConditionType.*Equal*;  
 }  
  
 public boolean isConditionGreaterThan() {  
 return conditionType == ConditionType.*GreaterThan*;  
 }  
  
 public boolean isConditionLessThan() {  
 return conditionType == ConditionType.*LessThan*;  
 }  
  
 public boolean hasOperation() {  
 return operationType != null;  
 }  
  
 public OperationType getOperationType() {  
 return operationType;  
 }  
  
 public boolean mayHappen(int variableValue){  
 switch (conditionType){  
 case *Equal*: return variableValue == conditionComparedValue;  
 case *GreaterThan*: return variableValue > conditionComparedValue;  
 case *LessThan*: return variableValue < conditionComparedValue;  
 }  
 throw new RuntimeException("Somehow the value is not equal to, greater than or less than O.o");  
 }  
  
 public enum OperationType{  
 *None*,  
 *Set*,  
 Increment,  
 Decrement  
 }  
  
 public enum ConditionType{  
 Equal,  
 GreaterThan,  
 LessThan  
 }  
}

package main;  
  
import main.metamodel.Machine;  
import main.metamodel.State;  
  
public class MachineInterpreter {  
  
 private Machine machine;  
 private State currentState;  
  
 public void run(Machine m) {  
 machine = m;  
 currentState = m.getInitialState();  
 }  
  
 public State getCurrentState() {  
 return currentState;  
 }  
  
 public void processEvent(String event) {  
 var transition = currentState.getTransitionByEvent(event);  
 if(transition == null) return;  
 currentState = machine.getState(transition.getTarget().getName());  
 var integers = machine.getIntegers();  
 var key = transition.getOperationVariableName();  
 switch (transition.getOperationType()){  
 case *Set*: integers.put(key, transition.getOperationValue()); break;  
 case *Increment*: integers.put(key, integers.get(key) + 1); break;  
 case *Decrement*: integers.put(key, integers.get(key) - 1); break;  
 }  
 }  
  
 public int getInteger(String variableName) {  
 return machine.getIntegers().get(variableName);  
 }  
  
}

package main;  
  
import main.metamodel.Machine;  
import main.metamodel.State;  
import main.metamodel.Transition;  
  
public class StateMachine {  
  
 private final Machine machine = new Machine();  
 private State currentState;  
 private Transition currentTransition;  
  
 public Machine build() {  
 machine.getStates()  
 .forEach(s -> s.getTransitions()  
 .forEach(t -> t.setTarget(machine.getState(t.getTargetName()))));  
 return machine;  
 }  
  
 public StateMachine state(String name) {  
 currentState = new State(name, machine);  
 machine.getStates().add(currentState);  
 return this;  
 }  
  
 public StateMachine initial() {  
 machine.setInitialState(currentState);  
 return this;  
 }  
  
 public StateMachine when(String eventName) {  
 currentTransition = new Transition(eventName);  
 currentState.getTransitions().add(currentTransition);  
 return this;  
 }  
  
 public StateMachine to(String stateName) {  
 currentTransition.setTargetName(stateName);  
 return this;  
 }  
  
 public StateMachine integer(String name) {  
 machine.addInteger(name);  
 return this;  
 }  
  
 public StateMachine set(String variableName, int value) {  
 currentTransition.setOperationType(Transition.OperationType.*Set*);  
 currentTransition.setOperationVariableName(variableName);  
 currentTransition.setOperationValue(value);  
 return this;  
 }  
  
 public StateMachine increment(String variableName) {  
 currentTransition.setOperationType(Transition.OperationType.*Increment*);  
 currentTransition.setOperationVariableName(variableName);  
 currentTransition.setOperationValue(1);  
 return this;  
 }  
  
 public StateMachine decrement(String variableName) {  
 currentTransition.setOperationType(Transition.OperationType.*Decrement*);  
 currentTransition.setOperationVariableName(variableName);  
 currentTransition.setOperationValue(1);  
 return this;  
 }  
  
 public StateMachine ifEquals(String variableName, int value) {  
 currentTransition.setConditionType(Transition.ConditionType.*Equal*);  
 currentTransition.setConditionalVariableName(variableName);  
 currentTransition.setConditionComparedValue(value);  
 return this;  
 }  
  
 public StateMachine ifGreaterThan(String variableName, int value) {  
 currentTransition.setConditionType(Transition.ConditionType.*GreaterThan*);  
 currentTransition.setConditionalVariableName(variableName);  
 currentTransition.setConditionComparedValue(value);  
 return this;  
 }  
  
 public StateMachine ifLessThan(String variableName, int value) {  
 currentTransition.setConditionType(Transition.ConditionType.*LessThan*);  
 currentTransition.setConditionalVariableName(variableName);  
 currentTransition.setConditionComparedValue(value);  
 return this;  
 }  
  
}