Software Code Generation

Software Engineering
Prof. Maged Elaasar

Learning objectives

- Learn about model driven development (MDD)
- Learn when to use software code generation
- Learn how to do software code generation

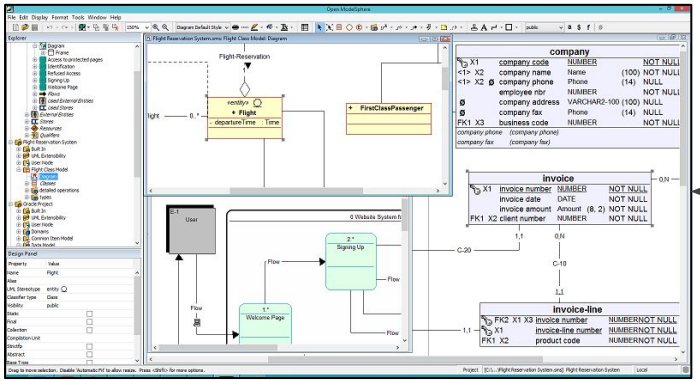
Model Driven Development

Modeling Myths

- Software modeling is for human **communication** only
- Software modeling can be done using **UML** only
- Software models are **notation** only
- Software modeling notation is **graphical** only
- Software models **generate worse code** than manually written one
- Software models are nothing more than **high level programming languages**

What is modeling driven development?

- MDD is a software development method that focuses on creating and exploiting models in software development activities
  - **Forward engineering**: translating models to code
  - **Reverse engineering**: translating code to models

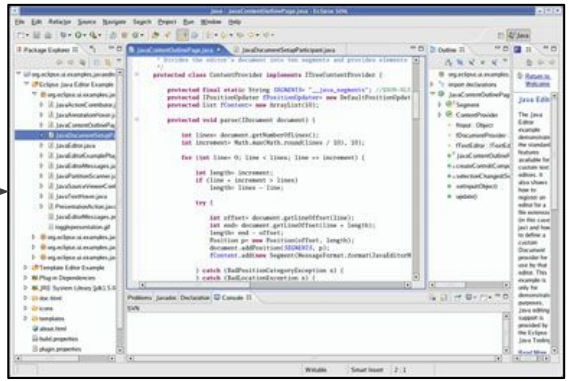


Model

Reverse engineering

Forward engineering

(code generation)

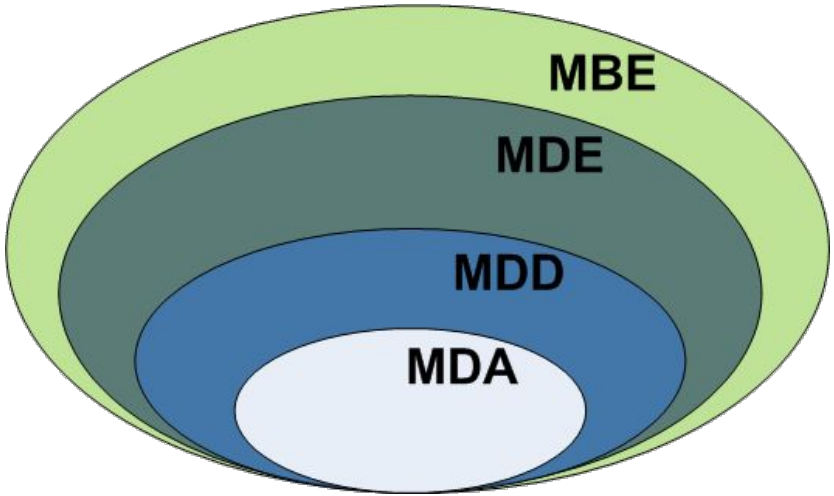


Code

MDD acronyms

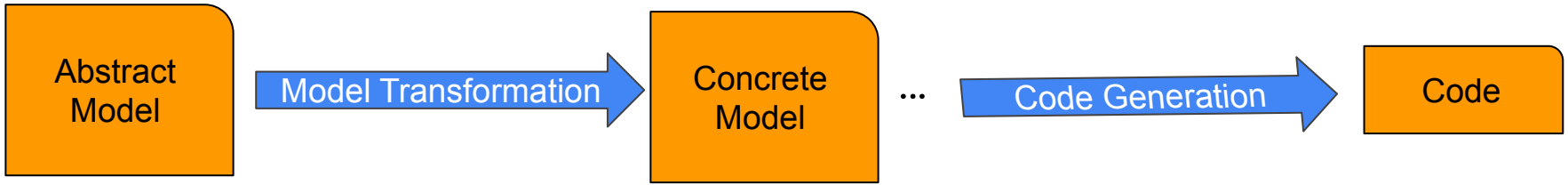
- **MBE**: Model Based Engineering (to analyze systems)
- **MDE**: Model Driven Engineering (to automate engineering activities)
- **MDD**: Model Driven Development (to automate software development)
- **MDA**: Model Driven Architecture (to practice MDD with the OMG standards)

Object Management Group
(a software standards body)



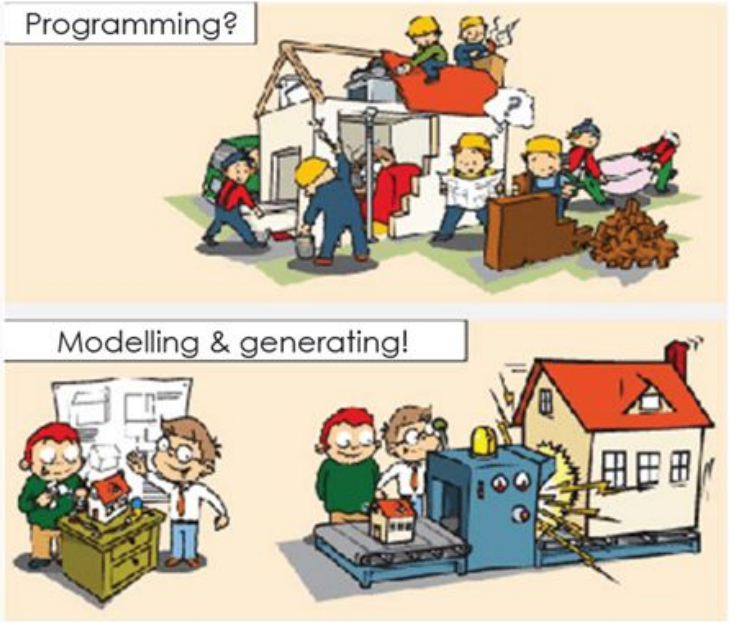
MDD Principles: Abstraction and Automation

- A model is an **abstraction** of a software that can be analyzed before the software is built and can be used to **automate** software development.
- A model is **abstracted** using a modeling language which can be
  - A generic modeling language (like UML)
  - A domain-specific language (DSL)
- A model is transformed **automatically** into a code using a transformation
  - A transformation can transform a model into a more concrete model first before code
  - The generated code can be templatized to fit the business domain



Major MDD Benefit: Productivity

- Write the code generator once and use it many times
- Specifying the model to the generator and invoking it is significantly **faster** than writing the code manually.



Other MDD Benefits

- Simplification
  - The source of truth becomes the model, not the code.
  - That model is easier to analyze than the generated code.
- Portability
  - Same model can generate code for different platforms (languages, frameworks, OSs, etc).
  - Same model can be used to generate different kinds of artifacts (code, db schema, spec, etc)
- Consistency
  - Generated code is typically more consistent than manually written code
  - Generated code can have desired principles, design patterns, and naming conventions

Issues with MDD

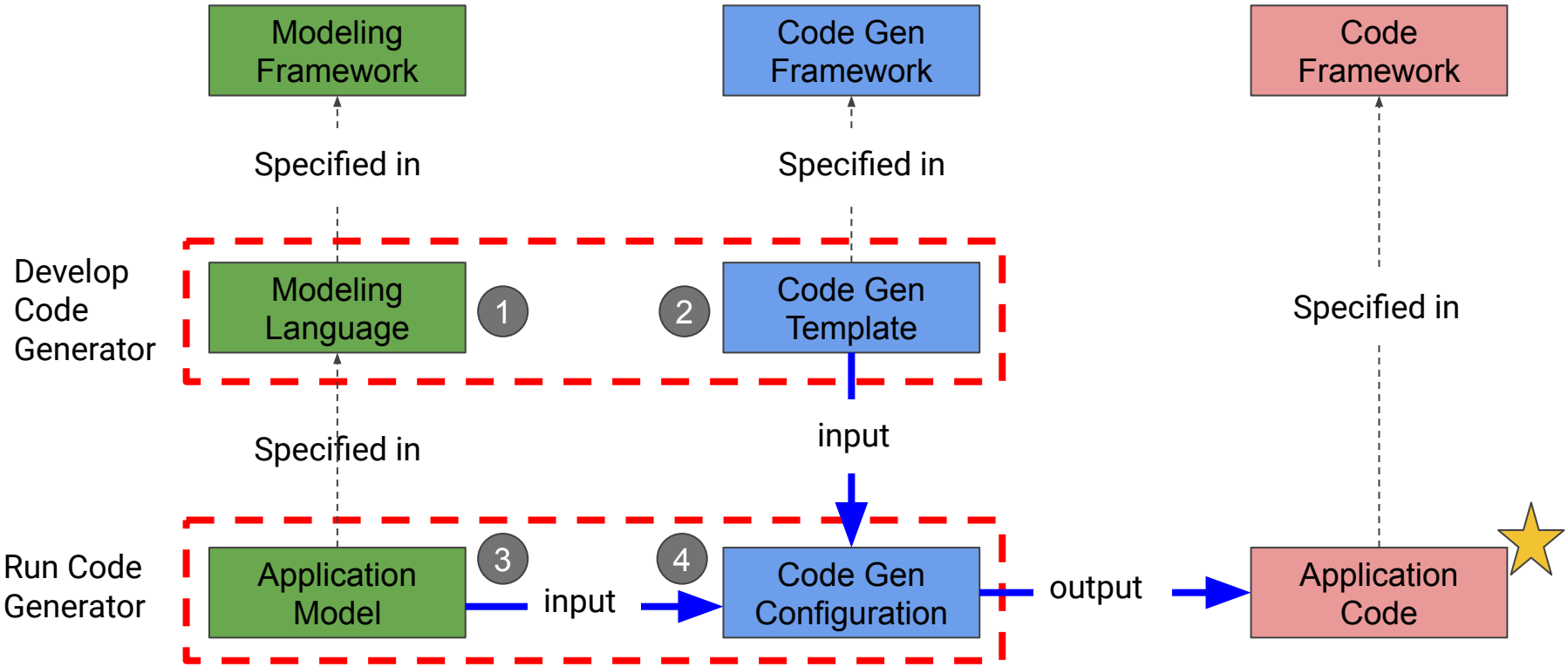
- Maintenance
  - Must have the right competency to develop a code generator.
  - Code generators must be maintained (fix bugs, adopt newer dependencies)
  - When you use a code generator tool your code becomes dependent on it

- Complexity
  - Generated code (if not customized) may be less optimized than the one you write by hand
  - Code templates may support more complex use cases than the code you need

MDD Quiz

Developing Code Generators

Code Generation Architecture



Steps to Develop a Code Generator

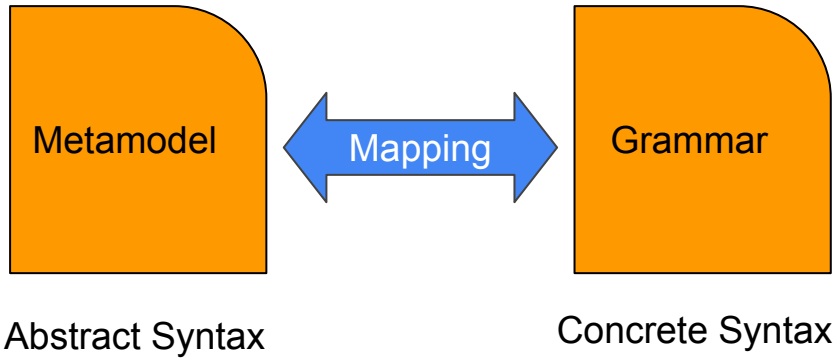
1. Develop a **modeling language** for the application domain
   a. Define the language syntax
   b. Generate the language API
   c. Use API to add validation rules

2. Develop a **code generation template** using the modeling language API

3. Develop an **application model** using the modeling language and run it through the code generation template to generate the **application code**

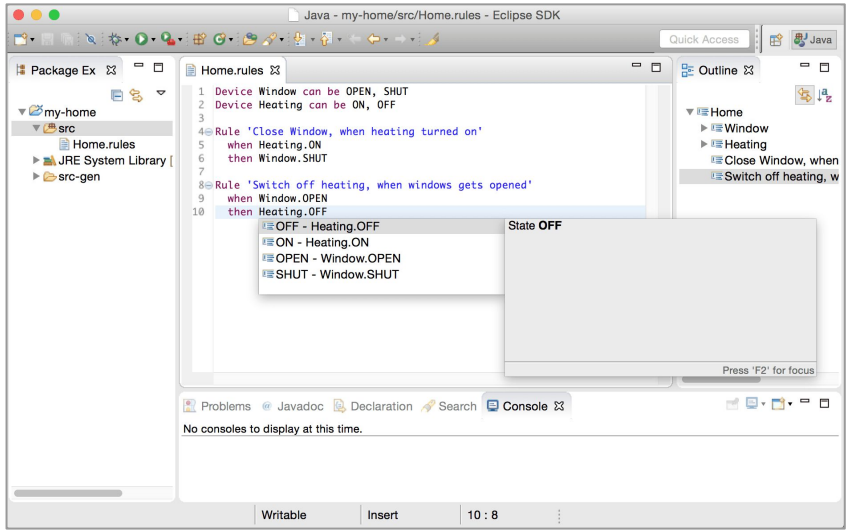
1. Define a Modeling Language

- A modeling language is defined using a modeling framework
  - **Abstract syntax**: defined with a class diagram (called a metamodel)
  - **Concrete syntax**: defined with a (textual and/or graphical) grammar



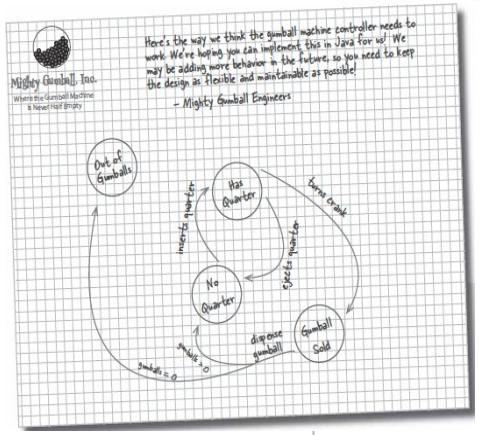
Modeling Framework

- **Xtext** is an open-source Java framework for defining textual modeling languages
- We will use Xtext to develop a **textual modeling language** for our code generator



https://www.eclipse.org/Xtext/

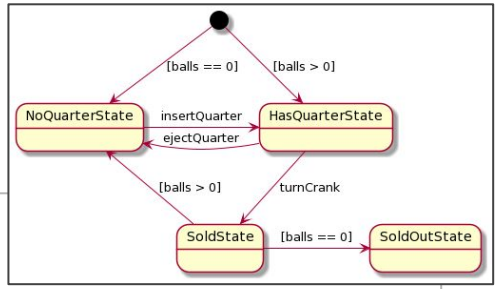
Example Language: Statemachine




```
state machine GumballMachine {
        initial start
        state NoQuarterState
        state HasQuarterState
        state SoldState
        state SoldOutState
        final end
        start -> NoQuarterState : none ["balls == 0"]
        start -> HasQuarterState : none ["balls > 0"]
        NoQuarterState -> HasQuarterState : insertQuarter
        HasQuarterState -> NoQuarterState : ejectQuarter
        HasQuarterState -> SoldState : turnCrank
        SoldState -> NoQuarterState : none ["balls > 0"]
        SoldState -> SoldOutState : none ["balls == 0"]
}
```

Code generator

```
@startuml
state NoQuarterState
state HasQuarterState
state SoldState
state SoldOutState
[*] -> NoQuarterState :  [balls == 0]
[*] -> HasQuarterState :  [balls > 0]
NoQuarterState -> HasQuarterState : insertQuarter
HasQuarterState -> NoQuarterState : ejectQuarter
HasQuarterState -> SoldState : turnCrank
SoldState -> NoQuarterState :  [balls > 0]
SoldState -> SoldOutState :  [balls == 0]
@enduml
```



Gumball.statemachine

Gumball.plantuml

Define Modeling Language BNF

- Xtext allows the definition of a modeling language's textual grammar and metamodel together using a form of BNF
- This is Statemachine.xtext

```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```

Grammar and Metamodel Names

- The grammar has a unique (qualified) name
- The metamodel has a name and a unique URI
- Other grammars can be imported and reused

```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```

grammar name

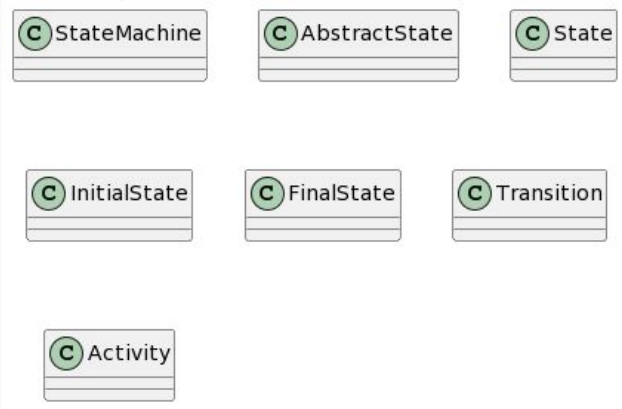
imported grammar

metamodel name and URI

Classes

- Xtext BNF is defined with named rules that represent the metamodel classes
- Rules have the syntax
  <name>:
  <expression>;
- Enumeration rules can be declared like this:
  enum <name>:
  <literal1>='<keyword1>'|
  <literal2>='<keyword2>' |
  <literal3>='<keyword3>';

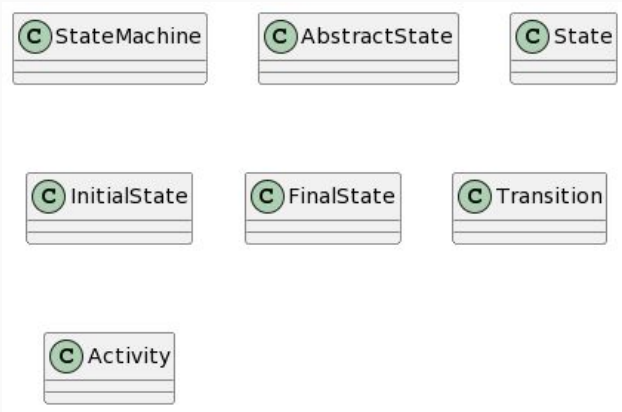
```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```



Keywords

- BNF rule expressions may contain terminal or nonterminal symbols
- Terminal symbols are enclosed within quotes and represent the grammar's keywords

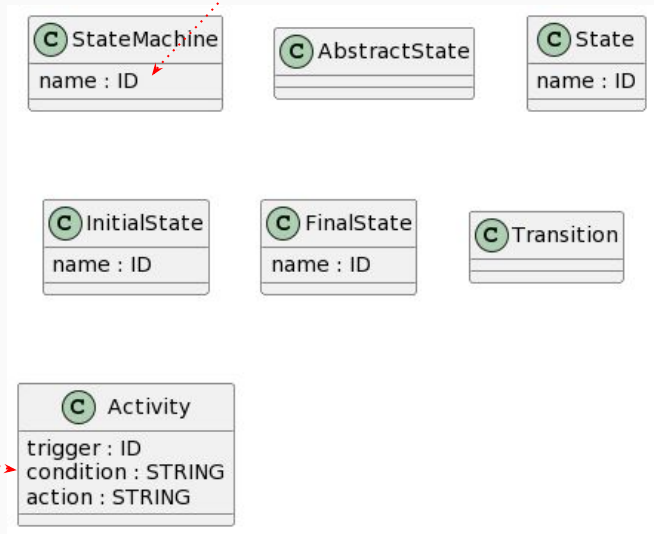
```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```



Attributes

- Nonterminal symbols represent class features
- Features have the syntax <name> = <type>
- When the <type> is a data type (ID, STRING, INTEGER, DOUBLE) the feature is an attribute of the rule's class
- Note that STRING is a quoted string (e.g., "hello there"), while ID is non-quoted string with no spaces (e.g., abc).
- A boolean attribute has the special syntax <name> ?= '<keyword>' (e.g., static ?= 'static')

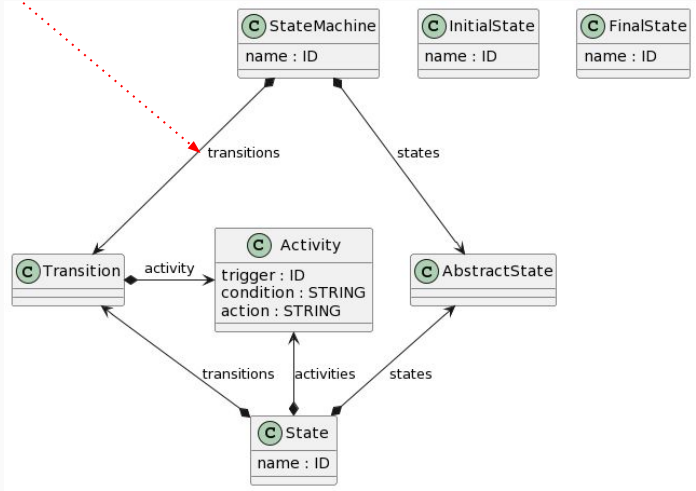
```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```



Compositions

- When a feature has the syntax <name> = <rule> then the feature represents a composition from the feature rule's class to the expression rule's class.

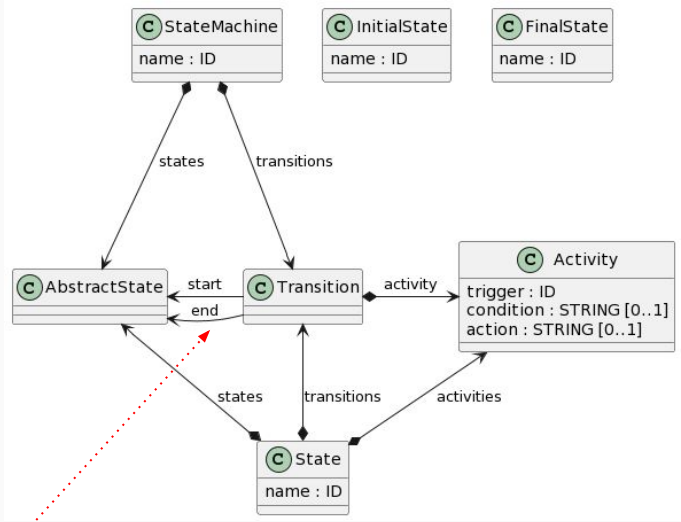
```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```



Associations

- When a feature has the syntax <name> = [<rule>] then the feature represents an association from the feature rule's class to the expression rule's class.
- [<rule>] represents a cross reference to an existing element of the rule's type
- An alternative syntax is [<rule>|ID] which makes it explicit that the cross reference is by ID.

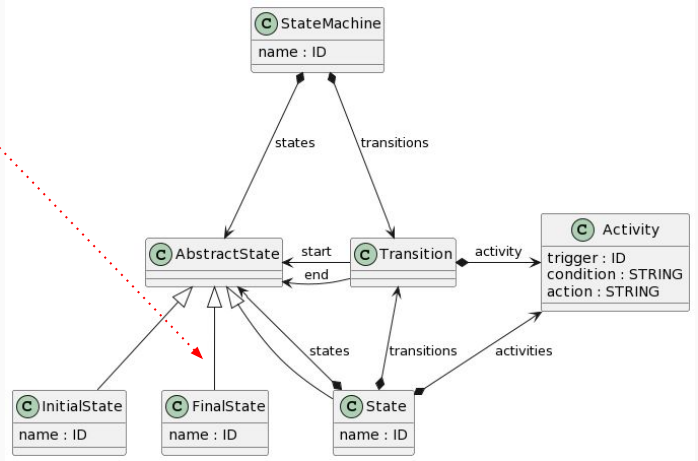
```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```



Inheritances

- Rules whose <expression> has the OR syntax: <rule> | <rule> .. | <rule> represent inheritance of the main rule by each of those ORed <rule>s

```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```

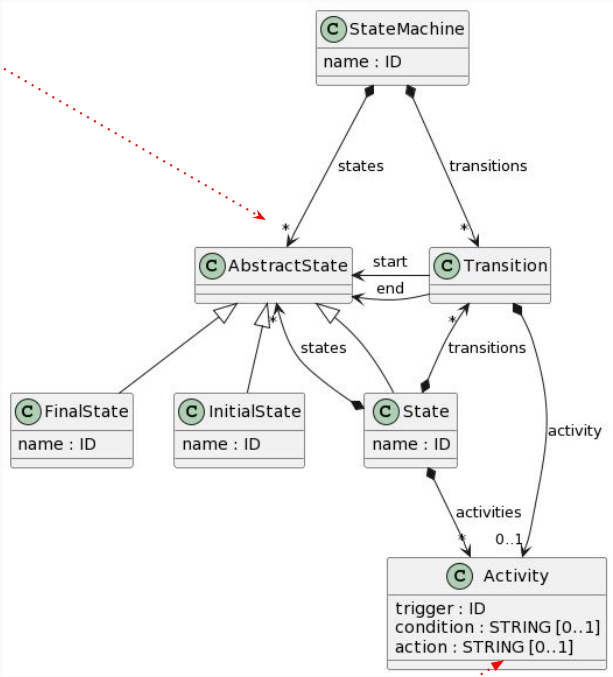


Cardinalities

- Default cardinality of a feature with the syntax <name>=<type> is [1].
- Other cardinalities can be expressed using the variants:

  <name>=<type>? is [0..1]
  <name>+=<type>+ is [1..*]
  <name>+=<type>* is [0..*]

```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```



Example Application Model: Gumball.statemachine

```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```

conforms to grammar

Gumball.statemachine

```
state machine GumballMachine {
        initial start
        state NoQuarterState
        state HasQuarterState
        state SoldState
        state SoldOutState
        final end
        start -> NoQuarterState : none ["balls == 0"]
        start -> HasQuarterState : none ["balls > 0"]
        NoQuarterState -> HasQuarterState : insertQuarter
        HasQuarterState -> NoQuarterState : ejectQuarter
        HasQuarterState -> SoldState : turnCrank
        SoldState -> NoQuarterState : none ["balls > 0"]
        SoldState -> SoldOutState : none ["balls == 0"]
}
```

Another Example Model
Example2.statemachine

```
grammar org.xtext.example.statemachine.Statemachine with org.eclipse.xtext.common.Terminals

generate statemachine "http://www.xtext.org/example/statemachine/Statemachine"

StateMachine:
        'state' 'machine' name=ID ('{'
                states+=AbstractState*
                transitions+=Transition*
        '}')?;

AbstractState:
        State | InitialState | FinalState;

State:
        'state' name=ID ('{'
                activities+=Activity*
                states+=AbstractState*
                transitions+=Transition*
         '}')?;

InitialState:
        'initial' name=ID;

FinalState:
        'final' name=ID;

Transition:
        start=[AbstractState] '->' end=[AbstractState] (':' activity=Activity)?;

Activity:
        trigger=ID ('[' condition=STRING ']')? ('/' action=STRING)?;
```

conforms to grammar

Example2.statemachine

```
state machine Example2 {
        initial I
        state S1
        state S2 {
                entry ["x<10"] / "x++"
                exit / "x--"
                initial I2
                state S21 {
                        entry / "y++"
                        exit / "y = y+2"
                }
                state S22
                final F2
                I2 -> S21 : e1 / "z++"
                S21 -> F2 : e3
                S22 -> F2 : e4
        }
        final F
        I -> S1
        S1 -> S2
        S2 -> F
}
```

At Home Exercise

- Draw the class diagram corresponding to this grammar
- Write an example model conforming to this grammar

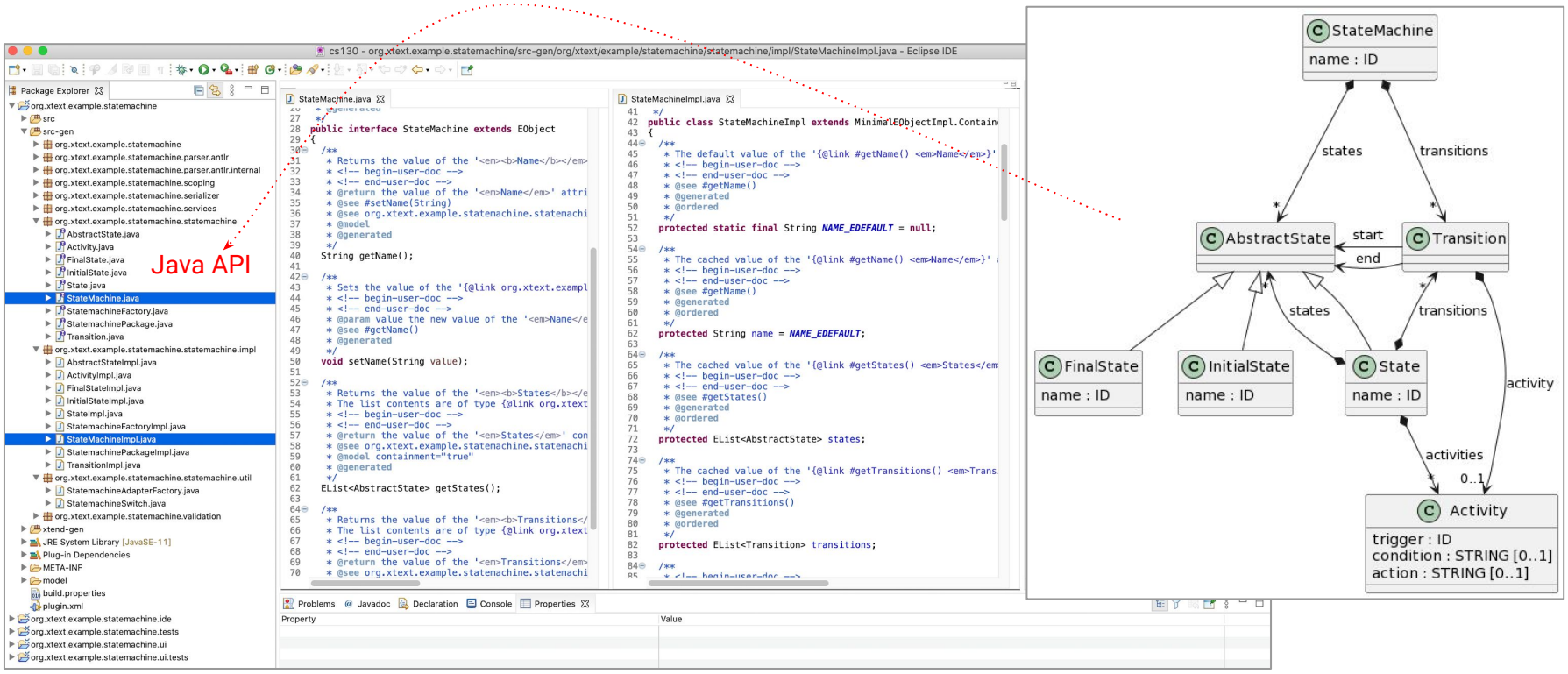
Note: feature cardinalities might be changed by surrounding them by parentheses with other cardinality symbols. E.g.:

feature=ID → [1..1]
(feature=ID)* → [0..*]
feature=ID+ → [1..*]
(feature=ID+)? → [0..*]

```
grammar org.xtext.example.SecretCompartments with org.eclipse.xtext.common.Terminals

generate secrets "http://www.eclipse.org/secretcompartment"

Statemachine :
 'events'
   events+=Event+
 'end'
 ('resetEvents'
   resetEvents+=[Event]+
 'end')?
 'commands'
   commands+=Command+
 'end'
 states+=State+;

Event :
 name=ID code=ID;

Command :
 name=ID code=ID;

State :
 'state' name=ID
   ('actions' '{' actions+=[Command]+ '}')?
   transitions+=Transition*
 'end';

Transition :
 event=[Event] '=>' state=[State];
```

Generate Modeling Language API

- Xtext generates Java API corresponding to the class diagram of the grammar

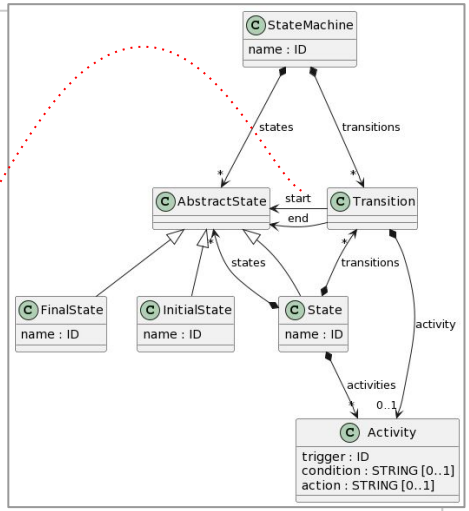


Develop Extra Validation Rules using API

- Xtext generates a Validator Java class for the language where validation extra rules can be added
  - Each rule is a function annotated with @check and has a single parameter typed by a class from language

```
package org.xtext.example.statemachine.validation;
import org.eclipse.xtext.validation.Check;
import org.xtext.example.statemachine.statemachine.*;
public class StatemachineValidator extends AbstractStatemachineValidator {
        public static final String TRANSITION_TO_INITIAL_STATE = "TransitionToInitialState";
        public static final String TRANSITION_FROM_INITIAL_STATE _TO_NON_LOCAL= "TransitionFromInitialStateToNonLocal";
        @Check
        public void checkTransitionToInitialState(Transition transition) {
                if (transition.getEnd() instanceof InitialState) {
                        error("Transition is not allowed to an initial state",
                                StatemachinePackage.Literals.TRANSITION__END,
                                RANSITION_TO_INITIAL_STATE);
                }
        }
        @Check
        public void checkTransitionFromNonLocalInitialState(Transition transition) {
                if (transition.getStart() instanceof InitialState) {
                        if (transition.getStart().eContainer() != transition.getEnd().eContainer()) { //eContainer() gets the composing (owning) element
                                error("Transition is not allowed from an initial state to a state in a different context",
                                        StatemachinePackage.Literals.TRANSITION__END,
                                        TRANSITION_FROM_INITIAL_STATE _TO_NON_LOCAL);
                        }
                }
        }
}
```

Rules on class Transition



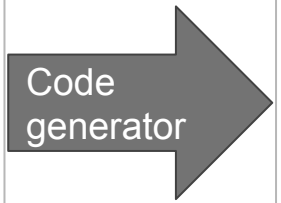
StatemachineValidator.java

2. Develop the Code Gen Transformation

Decide on the text you like to generate

```
state machine GumballMachine {
        initial start
        state NoQuarterState
        state HasQuarterState
        state SoldState
        state SoldOutState
        final end

        start -> NoQuarterState : none ["balls == 0"]
        start -> HasQuarterState : none ["balls > 0"]
        NoQuarterState -> HasQuarterState : insertQuarter
        HasQuarterState -> NoQuarterState : ejectQuarter
        HasQuarterState -> SoldState : turnCrank
        SoldState -> NoQuarterState : none ["balls > 0"]
        SoldState -> SoldOutState : none ["balls == 0"]
}
```

Gumball.statemachine




```
@startuml
state NoQuarterState
state HasQuarterState
state SoldState
state SoldOutState
[*] -> NoQuarterState :  [balls == 0]
[*] -> HasQuarterState :  [balls > 0]
NoQuarterState -> HasQuarterState : insertQuarter
HasQuarterState -> NoQuarterState : ejectQuarter
HasQuarterState -> SoldState : turnCrank
SoldState -> NoQuarterState :  [balls > 0]
SoldState -> SoldOutState :  [balls == 0]
@enduml
```

Gumball.plantuml

Develop the Code Gen Transformation

- Xtext allows a code generator to be developed with Xtend
- Xtend is a variant of Java that is more suited for developing code generators

```
package org.xtext.example.statemachine.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.AbstractGenerator
import org.eclipse.xtext.generator.IFileSystemAccess2
import org.eclipse.xtext.generator.IGeneratorContext

import org.xtext.example.statemachine.statemachine.*

class StatemachineGenerator extends AbstractGenerator {

   override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context)  {
           val machine = resource.contents.filter(StateMachine).head
           val text = generate(machine);
           val fileName = resource.URI.trimFileExtension.appendFileExtension("plantuml").lastSegment
           fsa.generateFile(fileName, text);
   }
   …// the rest on the next slide

}
```

StatemachineGenerator.xtend

https://www.eclipse.org/xtend/

Develop the Core Gen Transformation

This is the rest of the Xtend transformation code from the previous slide

```
def String generate(StateMachine machine) '''
        @startuml
        «FOR state : machine.states.filter(State)»
                «generate(state)»
        «ENDFOR»
        «FOR transition : machine.transitions»
                «generate(transition)»
        «ENDFOR»
        @enduml
'''
def String generate(State state) '''
        state «state.label»
        «IF !state.states.isEmpty() || !state.transitions.isEmpty() || !state.activities.isEmpty» {
                «FOR activity : state.activities»
                        «state.name» : «generate(activity)»
                «ENDFOR»
                «FOR nestedState : state.states.filter(State)»
                        «generate(nestedState)»
                «ENDFOR»
                «FOR transition : state.transitions»
                        «generate(transition)»
                «ENDFOR»
        }
        «ENDIF»
'''
```

Xtend template function starts and ends with three quotes

Xtend code has to be surrounded by « »

static text is unquoted and can be indented

```
def String generate(Transition transition) '''
        «transition.start.label» -> «transition.end.label»
        «IF transition.activity !== null» : «generate(transition.activity)»
        «ENDIF»
'''

def String generate(Activity activity) '''
        «IF activity.trigger != "none"»«activity.trigger»«ENDIF»
        «IF activity.condition !== null» [«activity.condition»] «ENDIF»
        «IF activity.action !== null» / «activity.action»«ENDIF»
'''

def String getLabel(AbstractState state) '''
        «IF state instanceof State»«state.name»«ELSE»[*]«ENDIF»'''
```

Same language API can be called from Xtend in the code ge template

StatemachineGenerator.xtend

3. Create Application Model and Generate Code

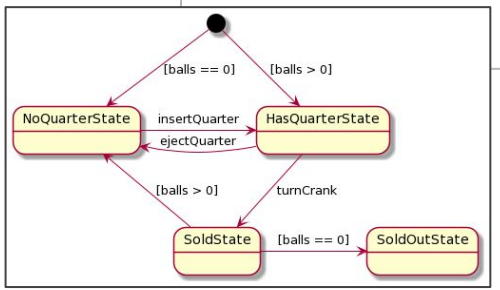
```
state machine GumballMachine {
        initial start
        state NoQuarterState
        state HasQuarterState
        state SoldState
        state SoldOutState
        final end
        start -> NoQuarterState : none ["balls == 0"]
        start -> HasQuarterState : none ["balls > 0"]
        NoQuarterState -> HasQuarterState : insertQuarter
        HasQuarterState -> NoQuarterState : ejectQuarter
        HasQuarterState -> SoldState : turnCrank
        SoldState -> NoQuarterState : none ["balls > 0"]
        SoldState -> SoldOutState : none ["balls == 0"]
}
```

Gumball.statemachine

Run Code generator

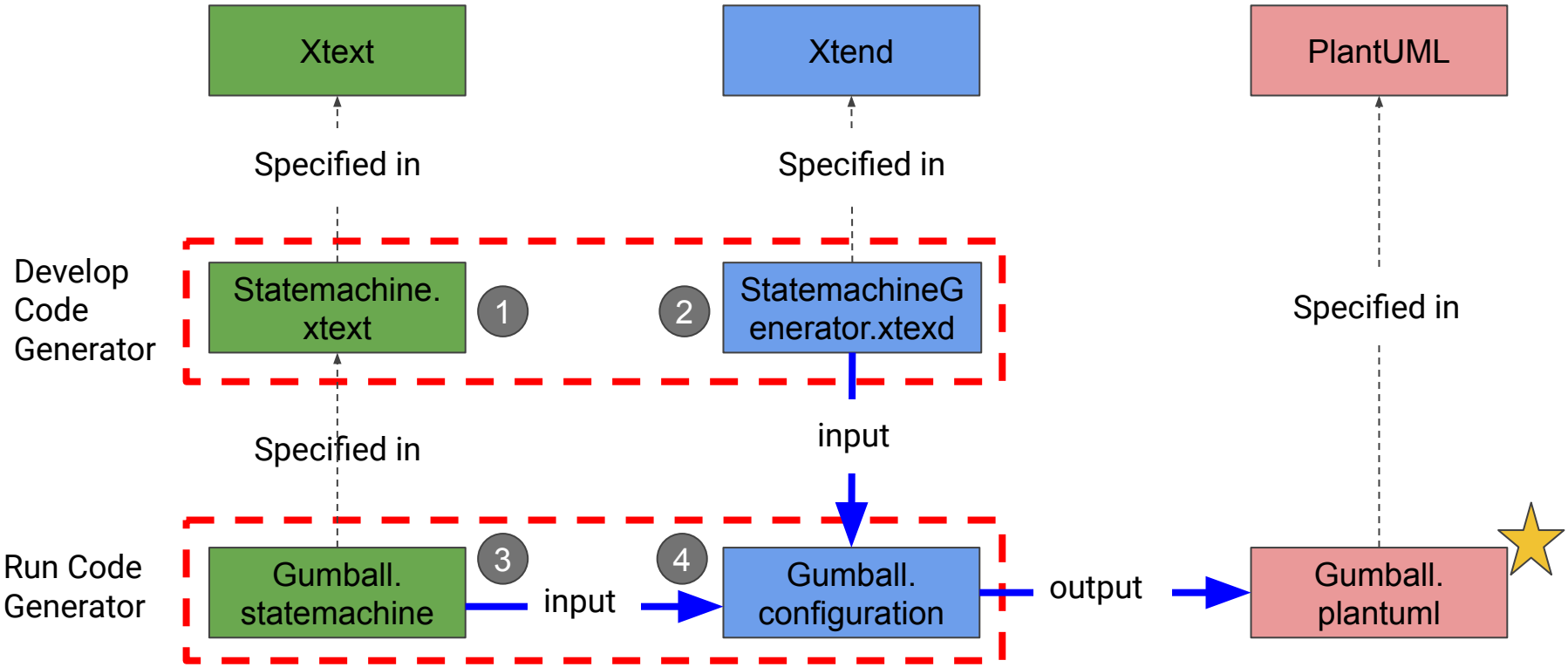
```
@startuml
state NoQuarterState
state HasQuarterState
state SoldState
state SoldOutState
[*] -> NoQuarterState :  [balls == 0]
[*] -> HasQuarterState :  [balls > 0]
NoQuarterState -> HasQuarterState : insertQuarter
HasQuarterState -> NoQuarterState : ejectQuarter
HasQuarterState -> SoldState : turnCrank
SoldState -> NoQuarterState :  [balls > 0]
SoldState -> SoldOutState :  [balls == 0]
@enduml
```

Gumball.plantuml



Rendered in plantUML

Example Code Gen Architecture



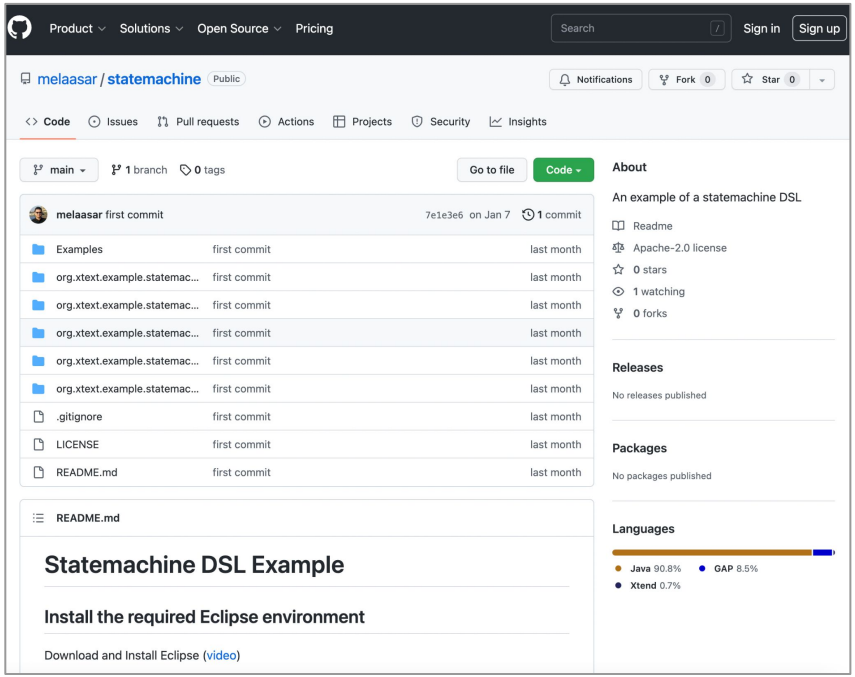
Xtext Tutorials

Step-by-Step Xtext Tutorials

- Xtext 5 minutes tutorial
  - https://www.eclipse.org/Xtext/documentation/101_five_minutes.html
- Xtext 15 minutes tutorial (basic)
  - https://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html
- Xtext 15 minutes tutorial (extended)
  - https://www.eclipse.org/Xtext/documentation/103_domainmodelnextsteps.html

Statemachine Tutorial with Videos

- The Statemachine code and video tutorial is provided in a github repository
  - https://github.com/melaasar/statemachine



References

- Xtext: Language Engineering for Everyone
  - https://www.eclipse.org/Xtext/documentation/index.html
- Xtend: Modernized Java
  - https://www.eclipse.org/xtend/documentation/index.html
- Scheneller, D.: “Code Generation with Xtext”, 2010.
  - http://www.danielschneller.com/2010/08/code-generation-with-xtext.html