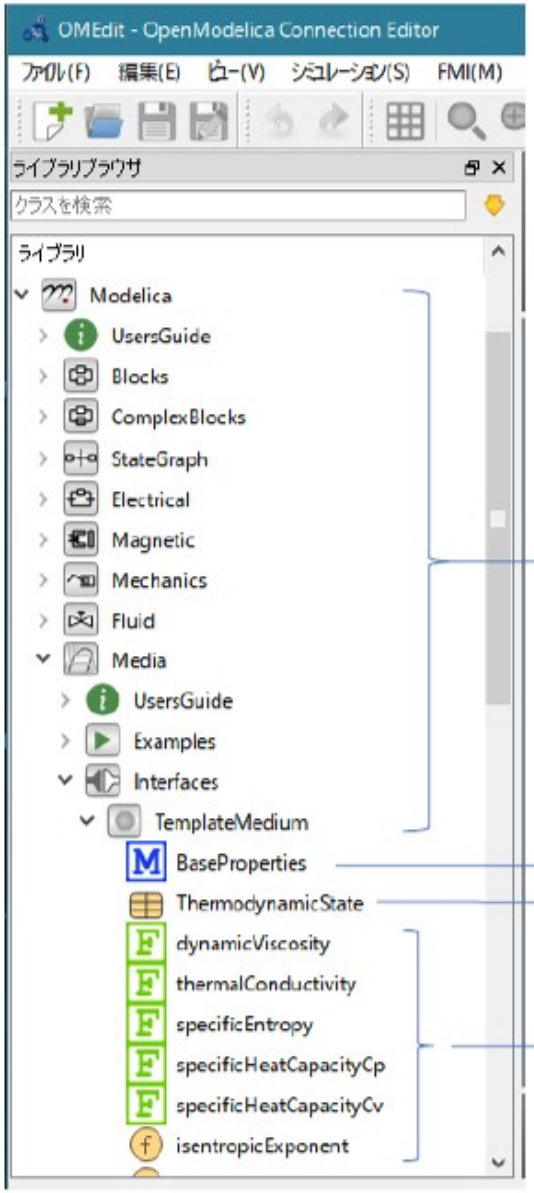
OpenModelica Intermediate course
Modelica.Fluid Library description

1. Modelica Class overview

December 7, 2017 Shu Tanaka (Amane Ryuken Co., Ltd.)

1. Modelica Class overview



OpenModelica https://www.openmodelica.org/
OMEdit Library browser

Libraries are made up of classes.

package

model

record

function

Class

Modelica classes
① package
② type
③ class
④ record
⑤ model
⑥ function
⑦ connector
⑧ block

Examples

First, an overview of features and use of classes that make up the library.

ClassExample1 Create a simple simulation model

Group variables using ClassExample2 record

Model using ClassExample3 block

Model using model with ClassExample4 connector

Try to inherit the ClassExample5 class

ClassExample6 Create a local class

ClassExample7 Create an replaceable local class

ClassExample8 Check physical properties of air using Modelica.Media

Examining room temperature changes using ClassExample9 Modelica.Media

ClassExample1

Create a simple simulation model

There are two ways to declare a class

A. Declaration method to assign an existing class

class Class name = base class name (...)

type, connector,...

```
type A = B(... );
```

B. Declaration method for listing components

class **List components between class name and end class name**

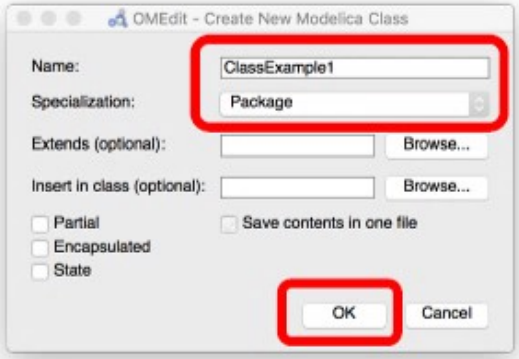
class, package, model,
block, connector,...

```
package C
  ...
end C;
```

```
class D
  ...
end D;
```

ClassExample1

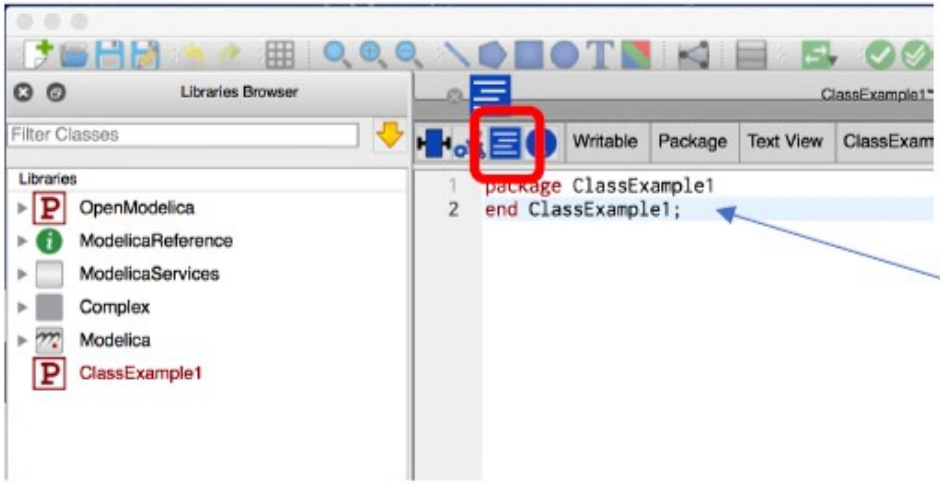
1. File > New Modelica Class



Name : ClassExample1

Class type: Package

2. Switch to Test View



3. Enter the code.

ClassExample1

```
package ClassExample1 // package    (1)
  constant Acceleration g = -9.8;

  // type
  type Acceleration = Real(quantity = "Acceleration", unit = "m/s2");   (2)
  type Mass = Real(quantity = "Mass", unit = "kg");                     (3)
  type Force = Real(quantity = "Force", unit = "N");                    (4)
  type Velocity = Real(quantity = "Velocity", unit = "m/s");            (5)
  type Position = Real(quantity = "Length", unit = "m");                (6)

  // class
  class MassA   (7)
    parameter Mass m = 1.0;
    parameter Force f = m*g;
    parameter Velocity v0 = 5.0;
    parameter Position x0 = 0.0;
    Position x(start = x0);
    Velocity v(start = v0);
  equation
    v = der(x);
    f = m*der(v);
  end MassA;

end ClassExample1;
```

Free-fall mass model

(Modelica.SIunits Is redeclared for explanation.)

Parameters

m: mass, f: external force,

x), v): Initial conditions

Variable x : Displacement v : Speed

Equation

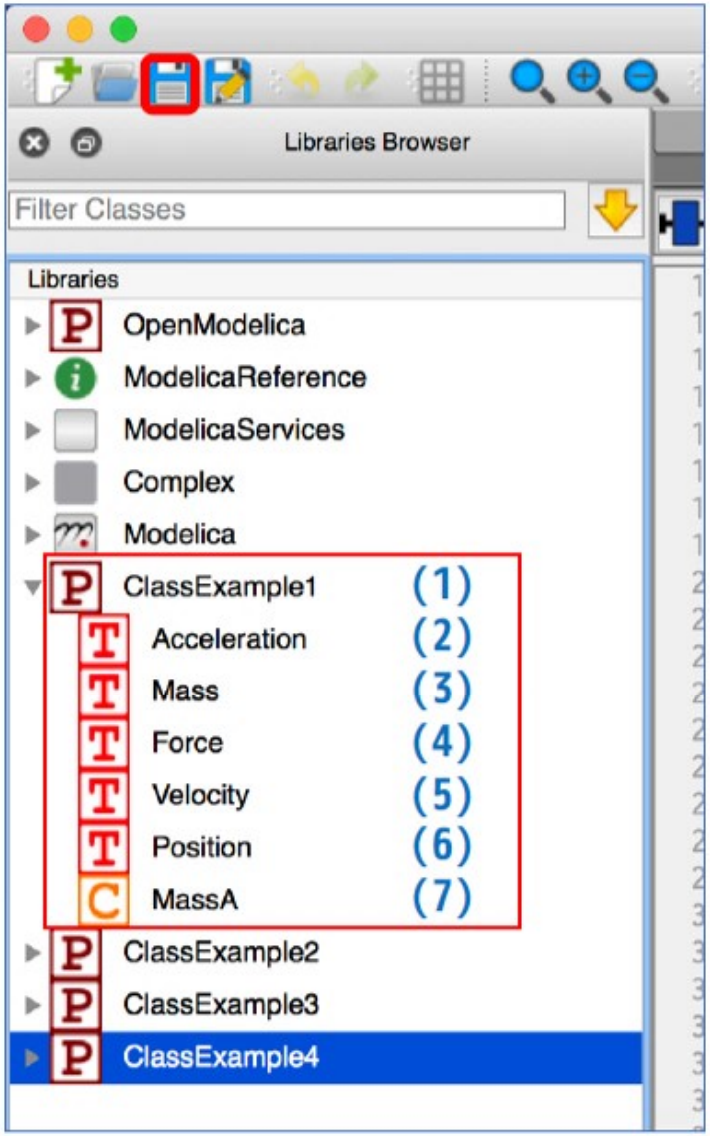
$$v = \frac{dx}{dt}, f = m\frac{dv}{dt}$$: Equation of the mass motion

Constitution element

Classes (1) to (7) were created.

ClassExample1

4. Save

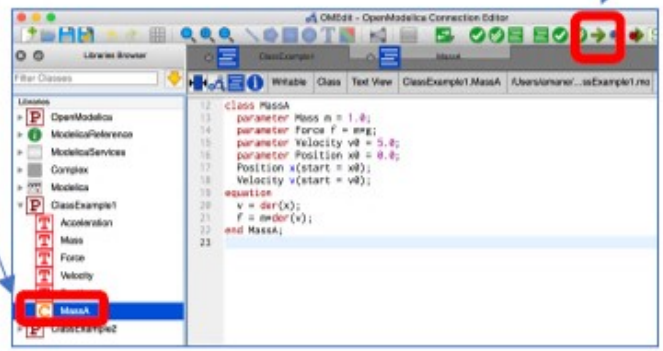


Library Browser

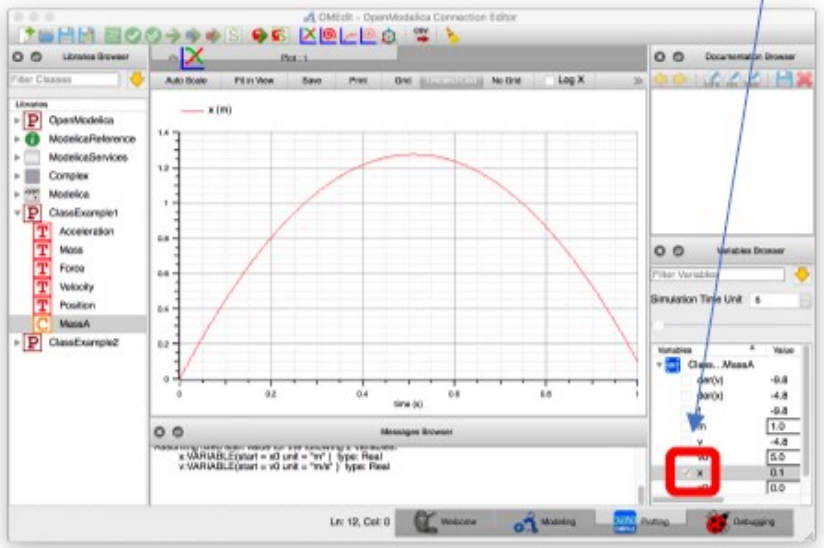
5. Execute

i. MassA
Double click

ii. Simulate
Click.

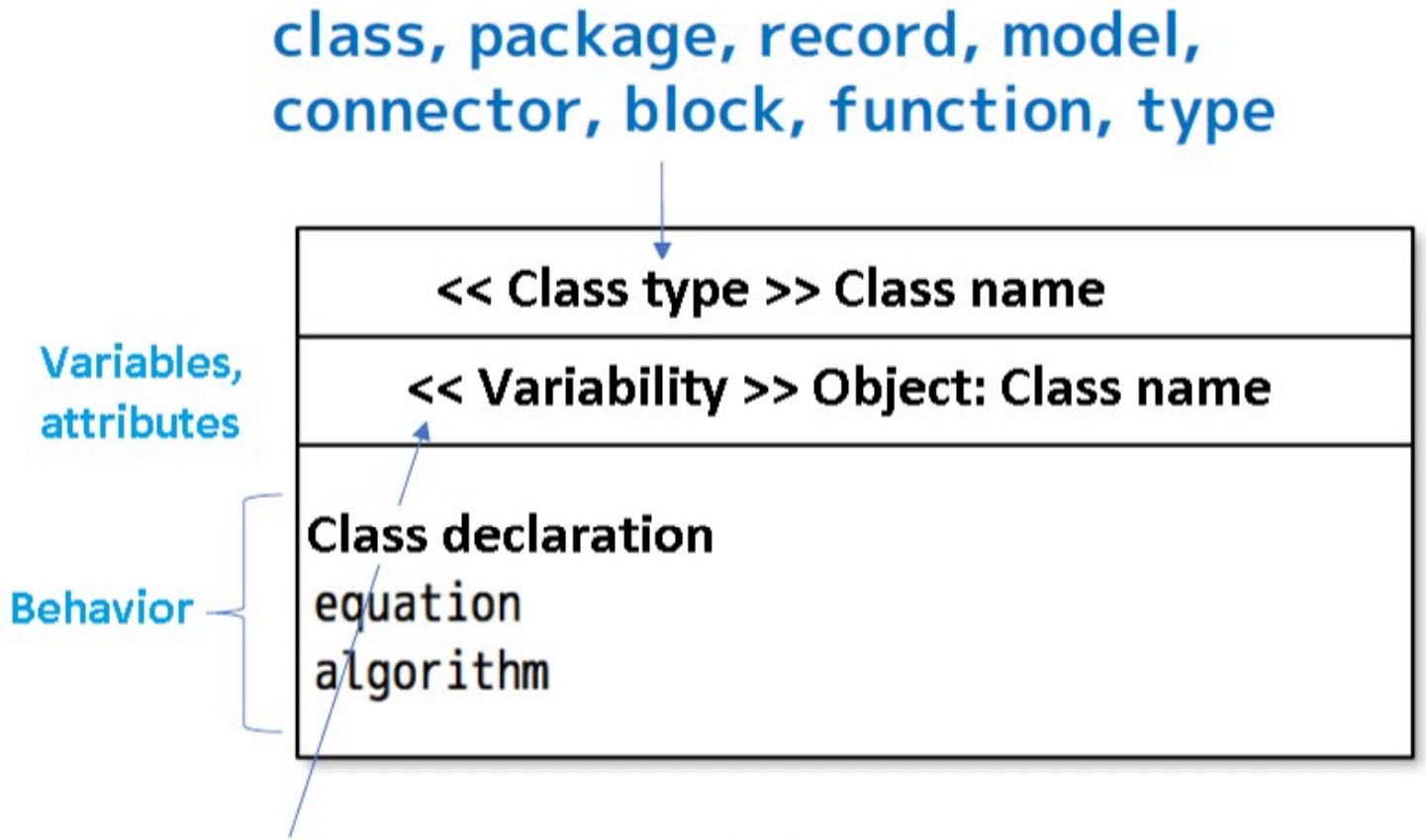


iii. Check x.



ClassExample1

In order to organize the features of Modelica classes, they are expressed in UML, and the features of the created classes are overviewed.



ClassExample1

① package (1)

- Only class declarations and constant definitions can be described inside.

```
<<package>> ClassExample1
<<constant>> g: Acceleration
type Acceleration
type Mass
type Force
type Velocity
type Position
class MassA
```

Constant

Class declaration

② type (2)~(6)

- predefined type and its array, can be declared only from a class that inherits the type.

```
type A = B(... );
```

predefined type

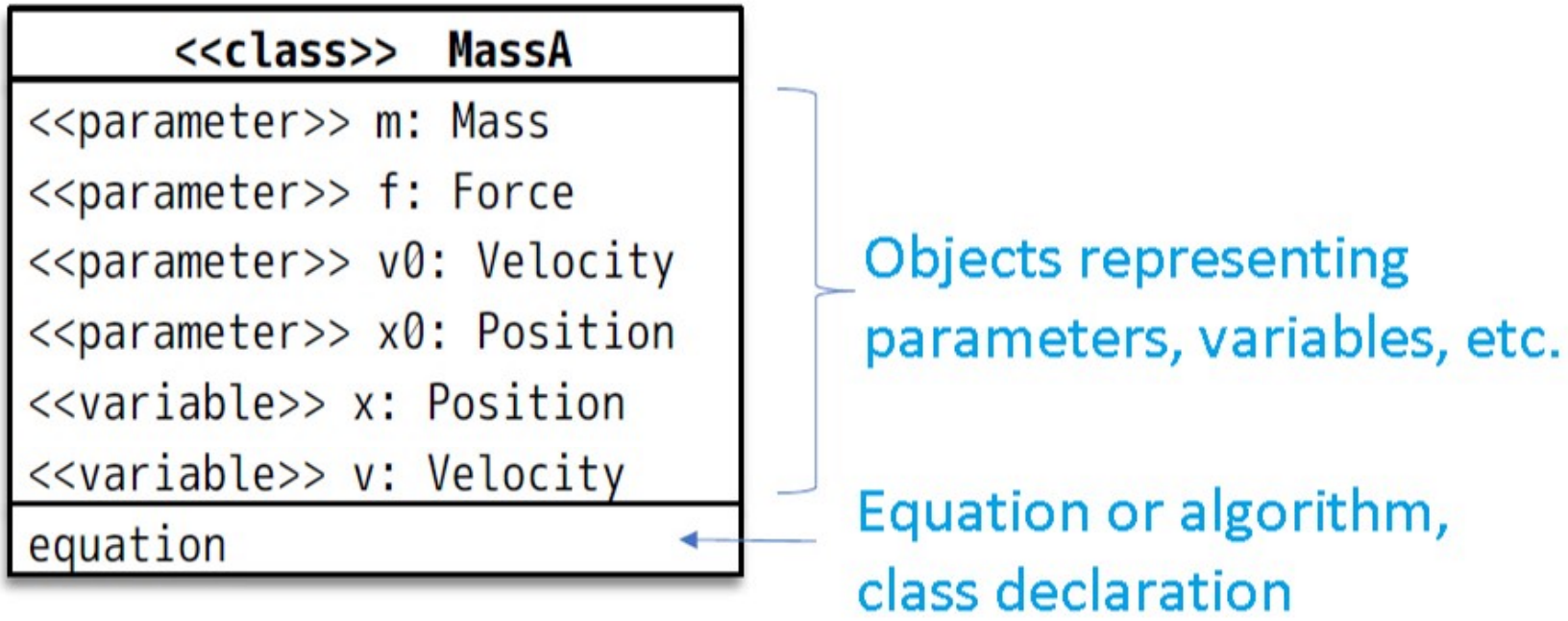
predefined type (built-in class)

- **Real**
- **Integer**
- **Boolean**
- **String**
- **enumeration(…)**

ClassExample1

③ class (Generic class) (7)

- Represent a generic class with no restrictions or enhancements. Same as model.
- In most cases, it is recommended to use a class that is specialized for a specific purpose, such as model or block, without using class.

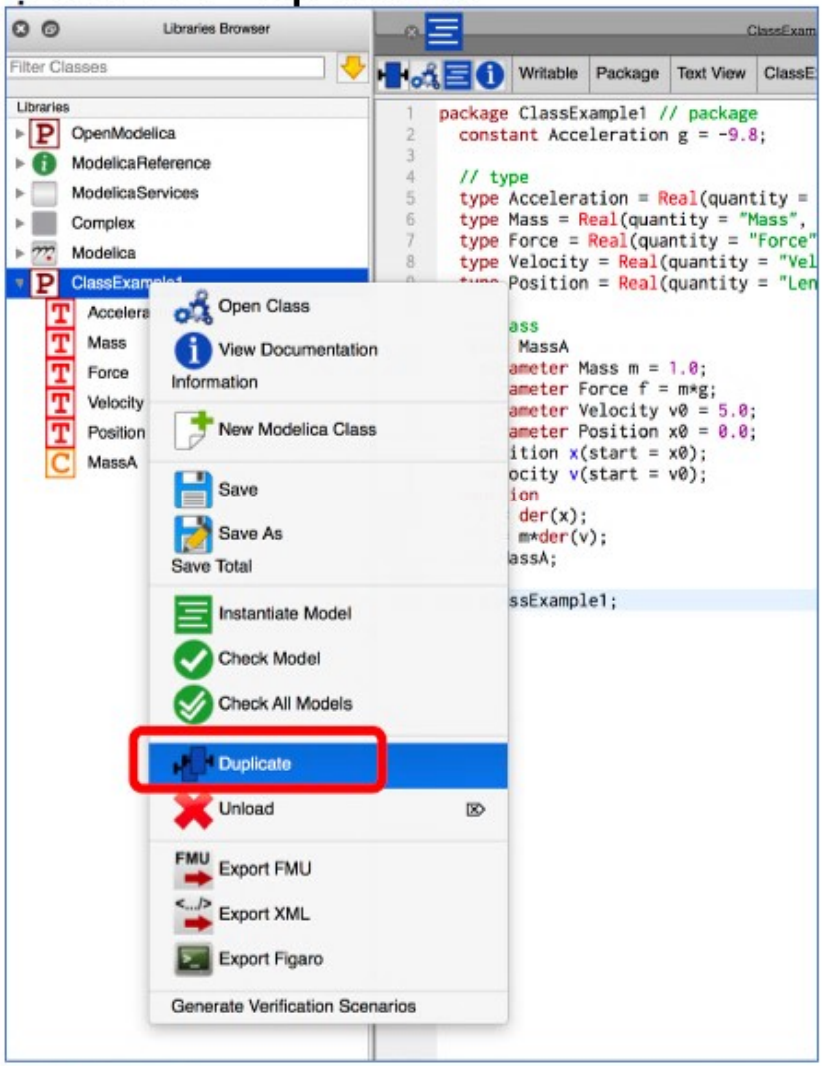


Simulation models can also be described

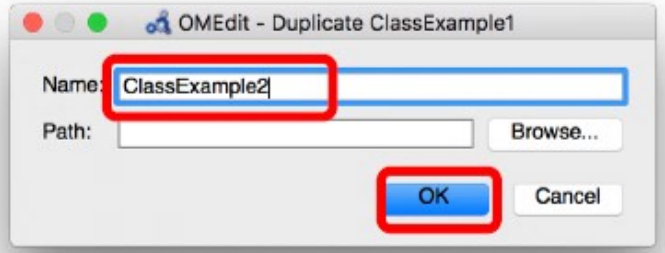
ClassExamle2

Group variables using record

1. Right-click ClassExample1 in the library browser
   Select Duplicate



2. Enter the class name.



3. Edit as in the next slide.

ClassExample2

```
package ClassExample2 // package

  constant Acceleration g = -9.8;

  // type
  type Acceleration = Real(quantity = "Acceleration", unit = "m/s2");
  type Mass = Real(quantity = "Mass", unit = "kg");
  type Force = Real(quantity = "Force", unit = "N");
  type Velocity = Real(quantity = "Velocity", unit = "m/s");
  type Position = Real(quantity = "Length", unit = "m");

  // record
  record State
    Position x;
    Velocity v;
  end State;

  // model
  model MassB
    parameter Mass m = 1.0;
    parameter Force f = m * g;
    parameter State s0(x = 0, v = 5);
    State s(x(start = s0.x), v(start = s0.v));
  equation
    s.v = der(s.x);
    f = m * der(s.v);
  end MassB;

end ClassExample2;
```

Record representing the motion state of the mass

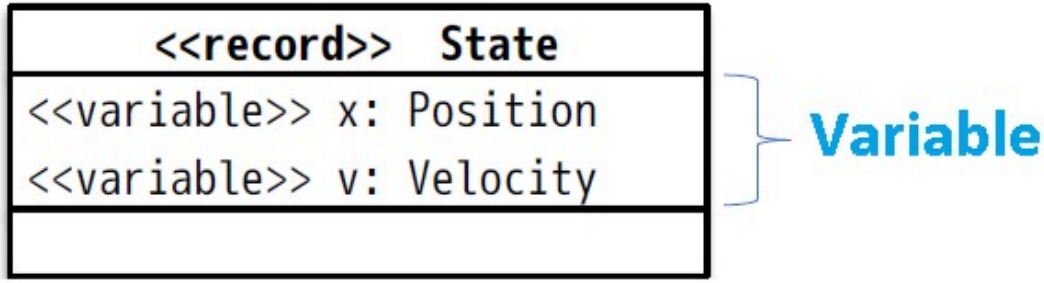
Use model instead of class.

Rewrite MassA using the state class object state as a state variable.

ClassExample2

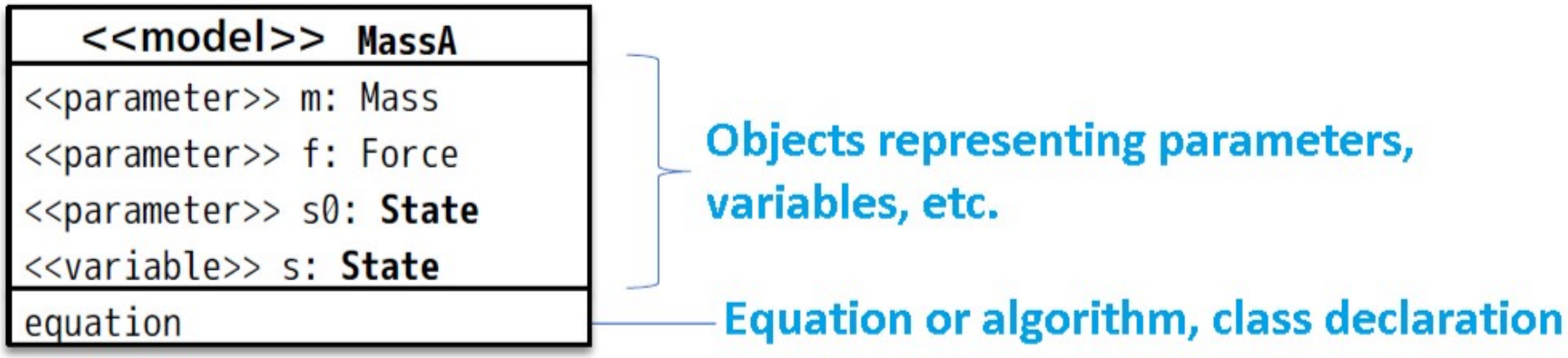
④ record

- Used to group variables.
- A prefix cannot be added in the variable definition.
- Do not describe behavioral equivalents.



⑤ model

- Primarily used to create simulation models by implementing parameters, variables, equations, and so on.
- Describe equation or algorithm.
- Effectively the same as class.

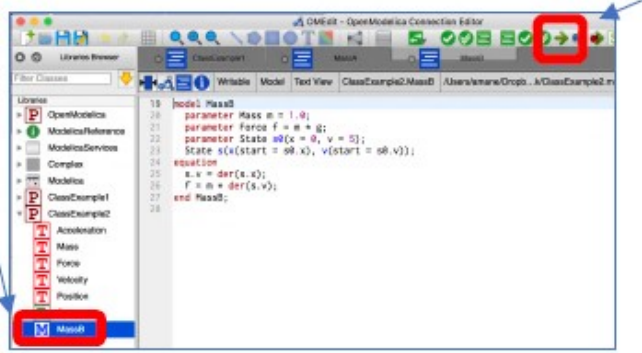


ClassExample2

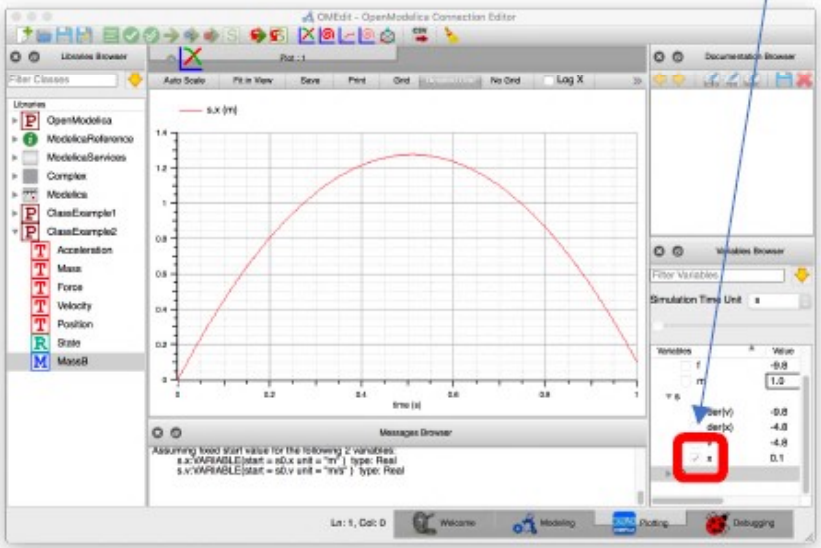
4. Run the simulation

i. Double-click MassB

ii. Click Simulate



iii. Expand s and check x.



ClassExamle3

Model using block

```
package ClassExample3  // package
  constant Acceleration g = -9.8;

  // type
  type Acceleration = Real(quantity = "Acceleration", unit = "m/s2");
  type Mass = Real(quantity = "Mass", unit = "kg");
  type Force = Real(quantity = "Force", unit = "N");
  type Velocity = Real(quantity = "Velocity", unit = "m/s");
  type Position = Real(quantity = "Length", unit = "m");
  type SpringConstant = Real(quantity = "Spring Constant", unit = "N/m");

  // record
  record State
    Position x;
    Velocity v;
  end State;

  // function
  function hookesLaw
    input SpringConstant k;
    input Position x;
    output Force f;
  algorithm
    f := -k*x;
  end hookesLaw;
```

ClassExample2 を
Duplicate and edit.

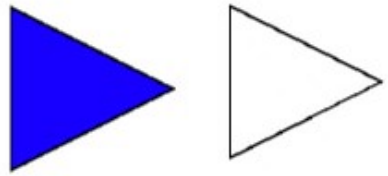
Function that represents
Hook's law

ClassExamle3

```
// connector(Redeclares what is in Modelica.Block.Interfaces.)
connector RealInput = input Real "'input Real' as connector" annotation( ...);
connector RealOutput = output Real "'output Real' as connector" annotation( ...);

// block
block MassBlock
   parameter Mass m = 1.0;
   parameter State s0(x = 0, v = 0);
   State s(x(start = s0.x), v(start = s0.v));
   RealInput f annotation( ...);
   RealOutput x annotation( ...);
equation
   s.v = der(s.x);
   f = m * der(s.v);
   x = s.x;
   annotation( ...);
end MassBlock;

block SpringBlock
   parameter SpringConstant k = 1.0;
   RealInput x annotation( ...);
   RealOutput f annotation( ...);
equation
   f = hookesLaw(k,x);
   annotation( ...);
end SpringBlock;
```

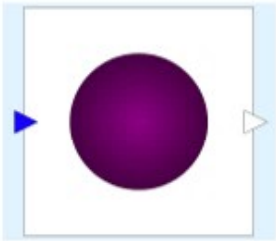
RealInput: Real input connector

RealOutput: real output connector



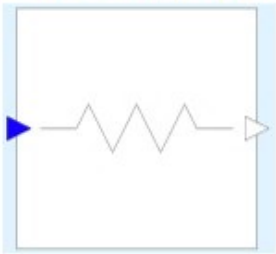
Block of mass moved by external force

Equation of motion



Spring block that generates elastic force against displacement

Use the function from the previous slide!



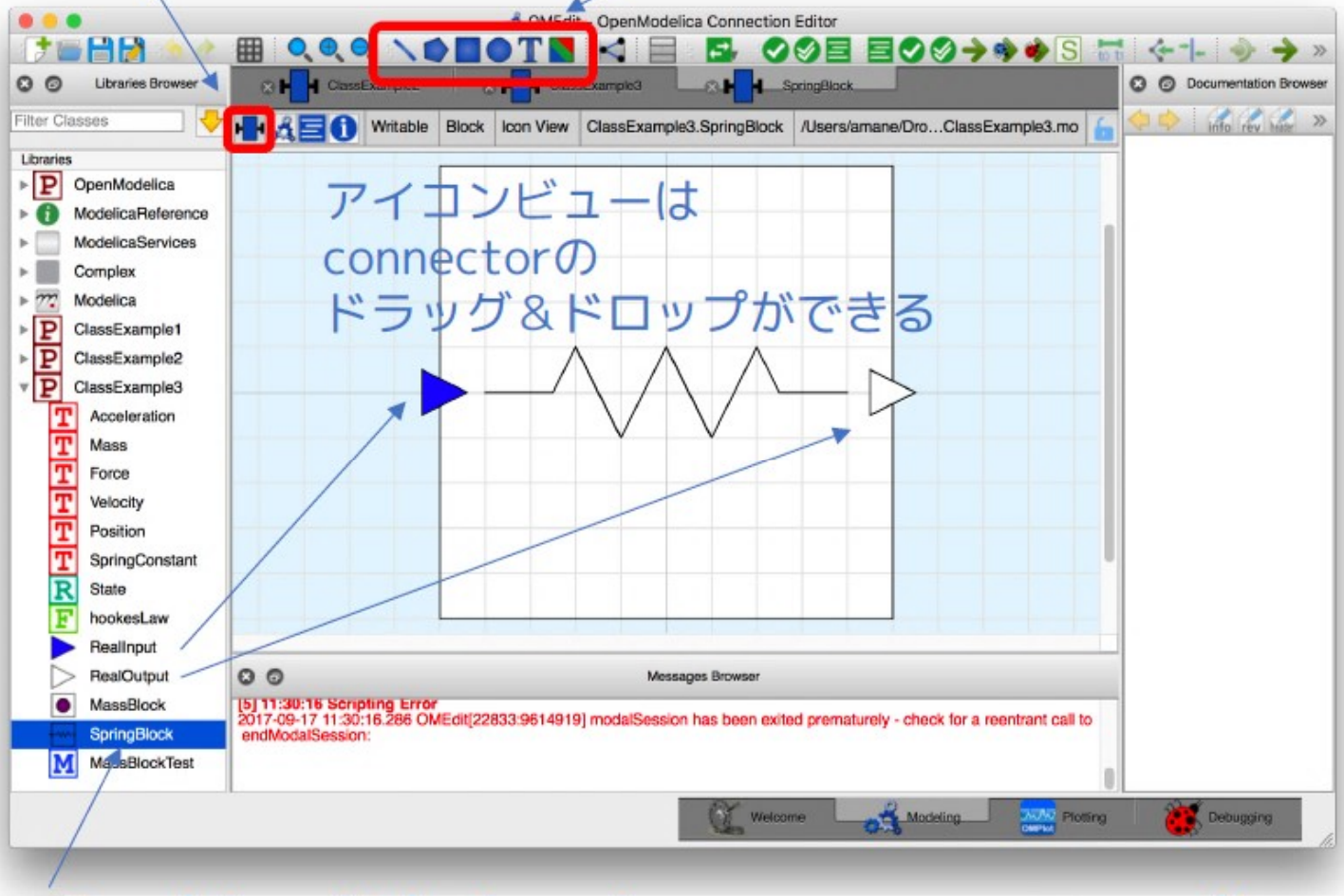
ClassExamle3

How to edit class icons

3. Draw

- Line
- Polygon
- Rectangle
- ellipse
- Text
- Image

2. Switch to Icon View



1.Double-click the class you want to edit to make it active.

ClassExamle3

⑥ function

- Used to define function
- Declare variables with input / output.
- Only algorithm statements can be defined.
- Function names are usually lowercase

<<function>> hookslaw
<<input variable>> k: SpringConstant <<input variable>> x: Position <<output variable>> f: Force
algorithm

Input variables

Output variables

Algorithm

⑦ connector (Simple one)

Connector to connect block and model

Similar to type but with a prefix

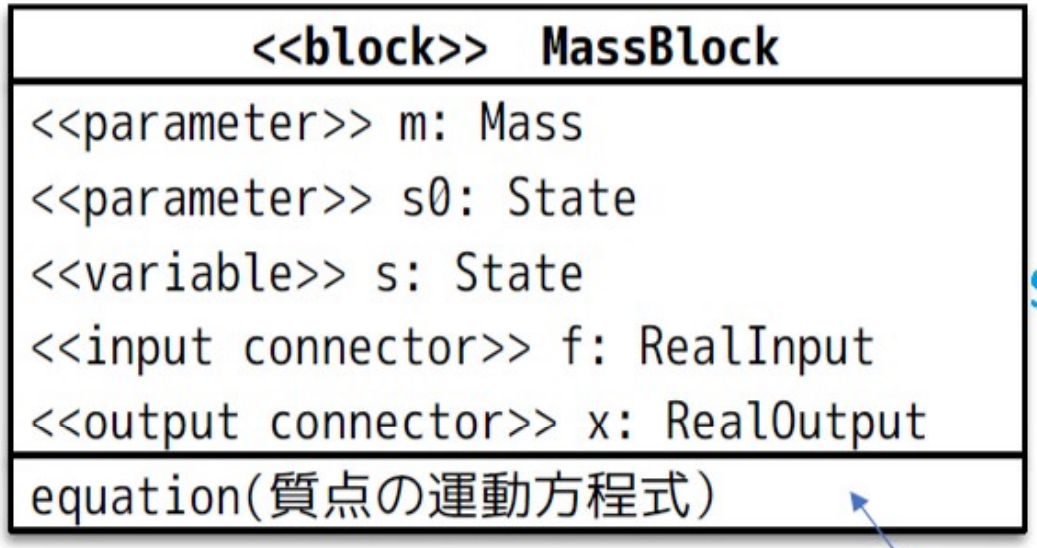
- Input Calculate values outside of class
- Output Calculate values within a class

```
connector A = input B;
```

ClassExamle3

⑧ block

- Almost the same as model, but always use a connector prefixed with input or output
- Used to create a block diagram



Parameters

State variables

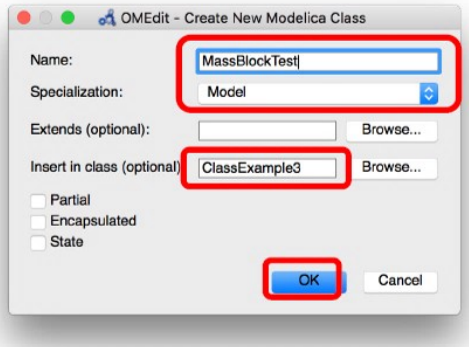
Input connector or output connector

Equation or algorithm

ClassExamle3

Create a test model for the block

1. File > New Modelica Class



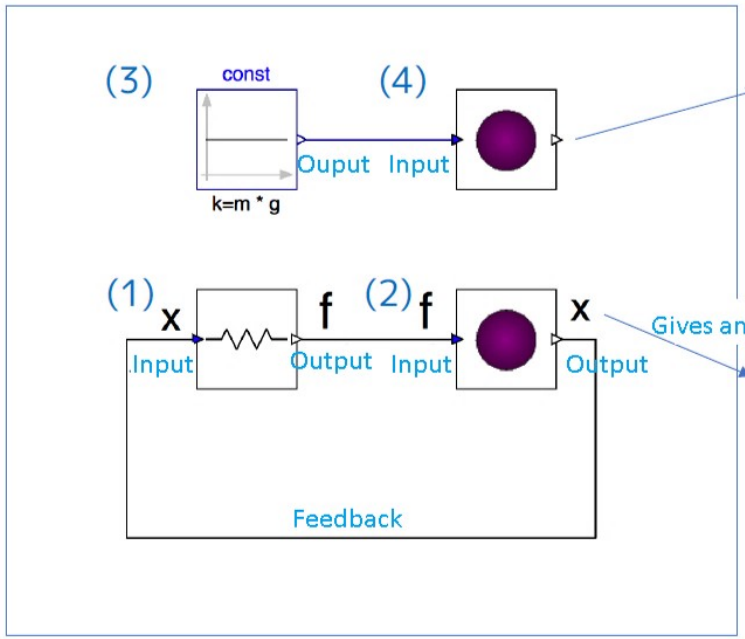
Name: MassBlockTest
Class type: Model
Class to insert: ClassExample3

2. Arrange and connect the blocks as shown in the next slide.

See the next slide source code for parameter settings!

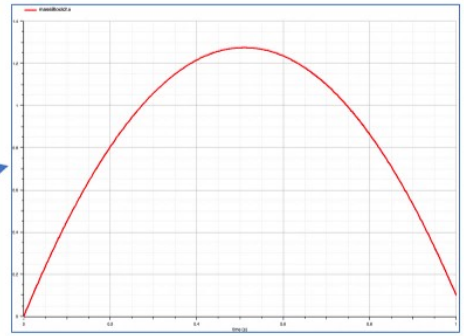
ClassExamle3

MassBlockTest

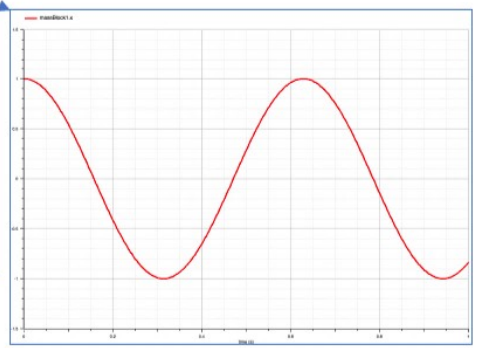


Model (block diagram)

Gravity is given to the mass point as an external force



Gives an elastic force to a mass point as an external force



Simulation result

ClassExamle3

MassBlockTest

```
// model
model MassBlockTest
    parameter Mass m = 1.0;
    SpringBlock springBlock1(k = 100) annotation( ...);                    (1)
    MassBlock massBlock1(m = m, s0(x = 1.0, v = 0.0)) annotation( ...);    (2)
    Modelica.Blocks.Sources.Constant const(k = m * g) annotation( ...);    (3)
    MassBlock massBlock2(m = m, s0(x = 0, v = 5.0)) annotation( ...);      (4)
equation
    connect(const.y, massBlock2.f) annotation( ...);
    connect(massBlock1.x, springBlock1.x) annotation( ...);
    connect(springBlock1.f, massBlock1.f) annotation( ...);
    annotation( ...);
end MassBlockTest;


end ClassExample3;
```

When connecting classes, the connection relation is defined as an equation using the operator connect ()

ClassExamle4

Model using model with connector

```
package ClassExample4  // package
  constant Acceleration g = -9.8;

  // type
  type Acceleration = Real(quantity = "Acceleration", unit = "m/s2");
  type Mass = Real(quantity = "Mass", unit = "kg");
  type Force = Real(quantity = "Force", unit = "N");
  type Velocity = Real(quantity = "Velocity", unit = "m/s");
  type Position = Real(quantity = "Length", unit = "m");
  type SpringConstant = Real(quantity = "Spring Constant", unit = "N/m");

  // record
  record State
    Position x;
    Velocity v;
  end State;

  // connector
  connector Flange
    Position s "Absolute position of flange";
    flow Force f "Cut force directed into flange";
    annotation( ...);
  end Flange;
```

Duplicate and edit ClassExample2.

Flange: Connector to connect 1D translation model

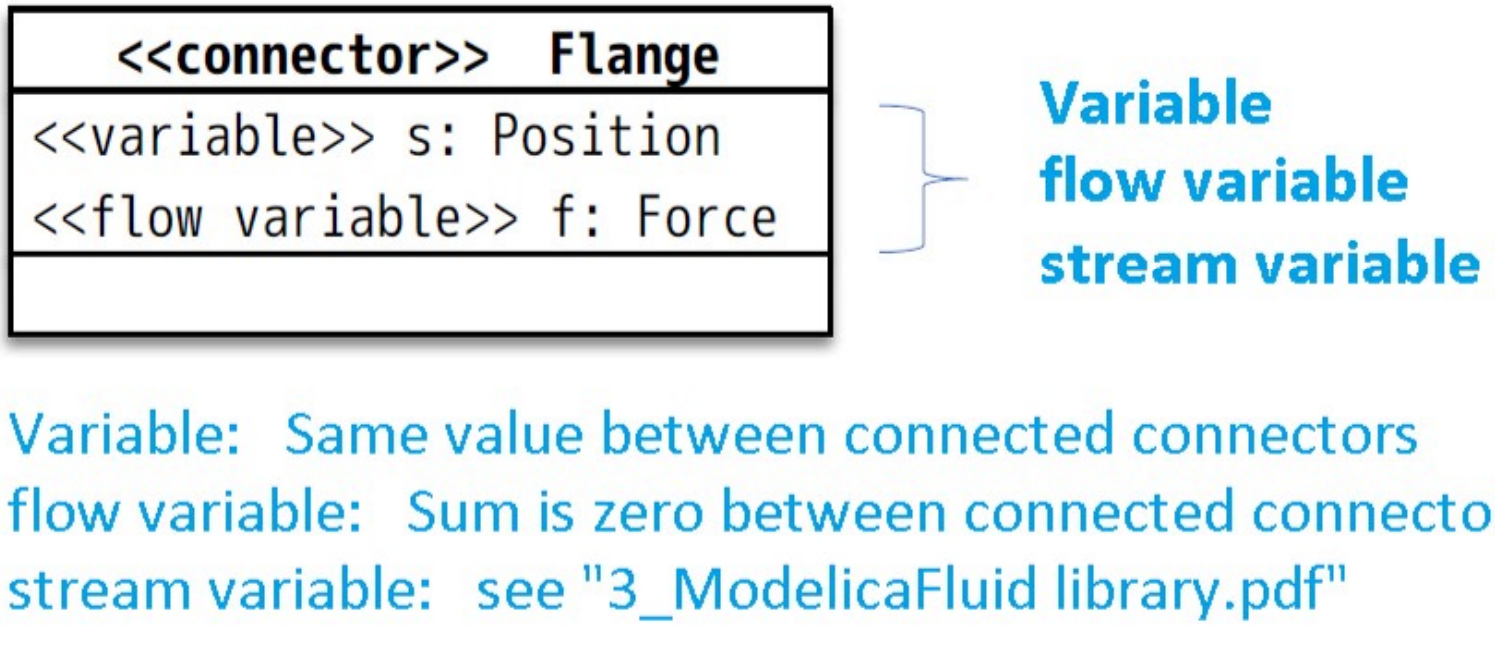
(Re-declared the same as Modelica.Mechanics.Translational.Interfaces.Flange_a.)

Between connected connectors
- Position s has the same value.
- The sum of force f becomes zero.

ClassExamle4

⑦ connector

- Similar to record, but represents a connector for connecting classes
- Equation and algorithm cannot be defined
- Prefixes such as flow and stream can be used in variable definitions
- Prefixes “input” and “output” can be attached to input and output



Variable: Same value between connected connectors
flow variable: Sum is zero between connected connectors
stream variable: see "3_ModelicaFluid library.pdf"

ClassExamle4

```
// model
 model MassModel
   parameter Mass m = 1.0;
   parameter State s0(x = 0, v = 0);
   State s(x(start = s0.x), v(start = s0.v));
   Flange flange annotation( ...);
 equation
   s.v = der(s.x);
   flange.f = m * der(s.v);
   flange.s = s.x;
   annotation( ...);
 end MassModel;

 model SpringModel
   parameter SpringConstant k = 1.0;
   Flange flange annotation( ...);
 equation
   flange.f = k * flange.s;
   annotation( ⋯);
 end SpringModel;
```

Create a model using the Flange connector variable.

Implement equation of motion

Mass model

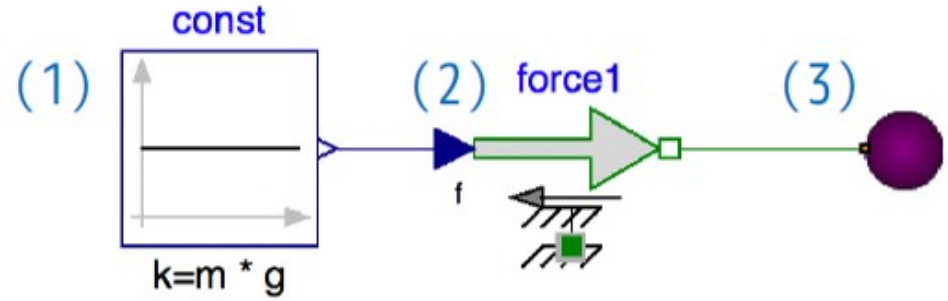
Implemented elastic force calculation formula

Spring model

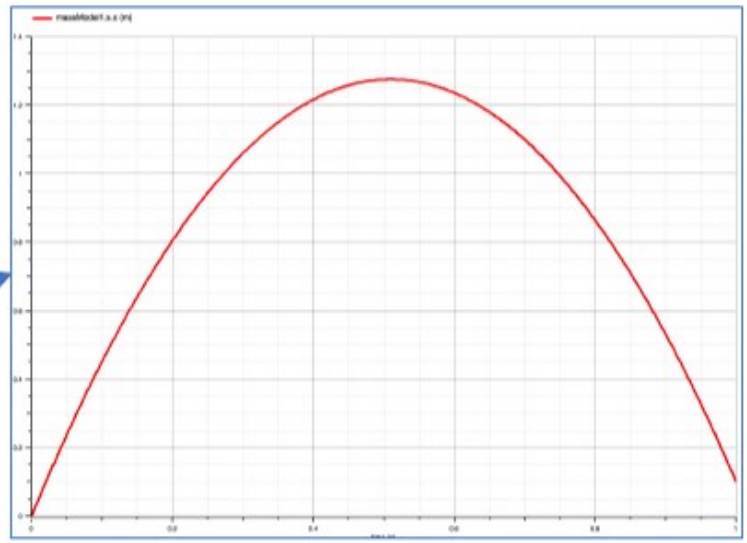
ClassExamle4

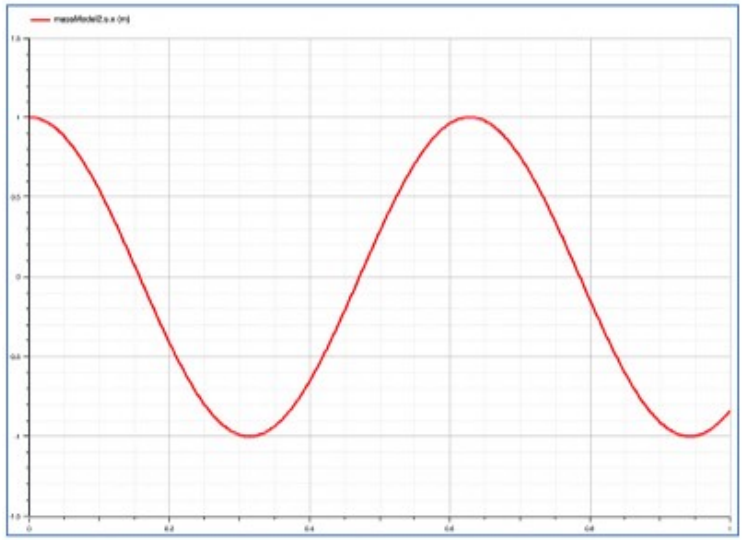
MassModelTest model for testing MassModel and SpringModel

Connect Force component
to mass to give gravity



Connect spring to mass point





Simulation result

ClassExamle4

MassModelTest model for testing MassModel and SpringModel

```
model MassModelTest
   parameter Mass m = 1.0;
   Modelica.Blocks.Sources.Constant const(k = m * g)  annotation( ...);          (1)
   Modelica.Mechanics.Translational.Sources.Force force1 annotation( ...);       (2)
   MassModel massModel1(m = m, s0(x = 0.0, v = 5.0)) annotation( ...);           (3)
   SpringModel springModel1(k = 100)  annotation( ...);                          (4)
   MassModel massModel2(m = m, s0(x = 1.0, v = 0.0))  annotation( ...);          (5)
equation
   connect(springModel1.flange, massModel2.flange) annotation( ...);
   connect(const.y, force1.f) annotation( ...);
   connect(force1.flange, massModel1.flange) annotation( ...);
end MassModelTest;

end ClassExample4;
```

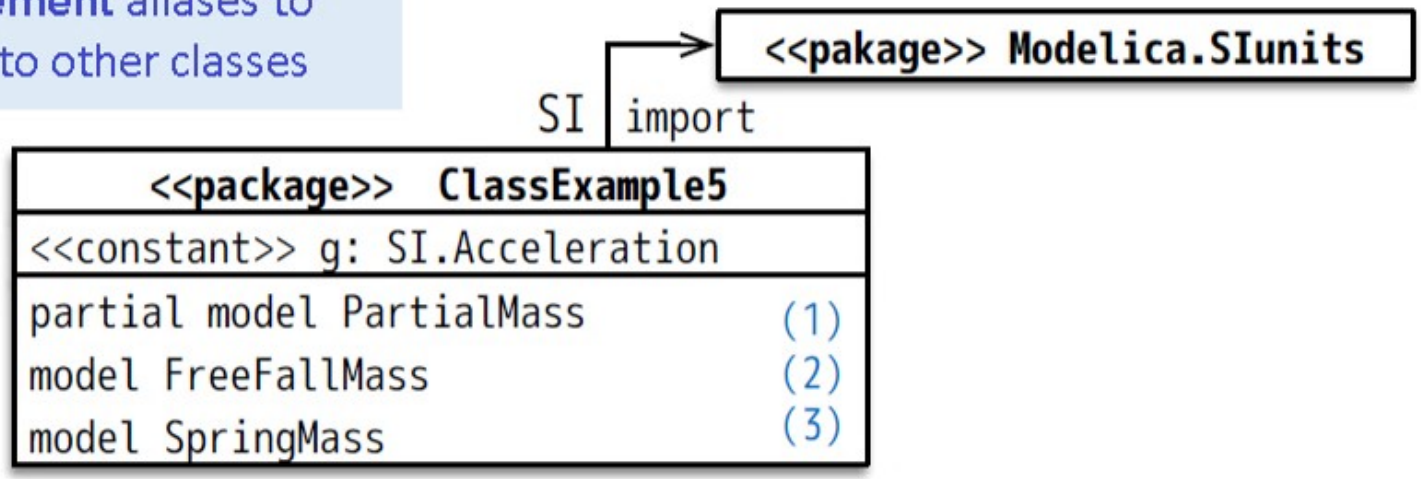
ClassExamle5

Try to inherit the class

A model with block or connector associates classes by connecting connectors.
There are other ways to associate classes.

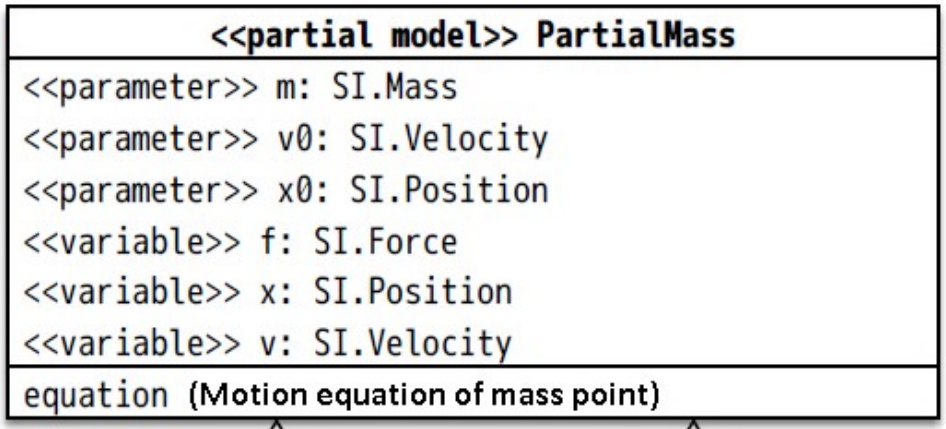
- Reference by alias (import statement)
- inheritance and expansion (extends, partial class)
- Local class
- Replaceable local classes (replaceable, redeclare)

Create **import statement** aliases to simplify references to other classes



ClassExamle5

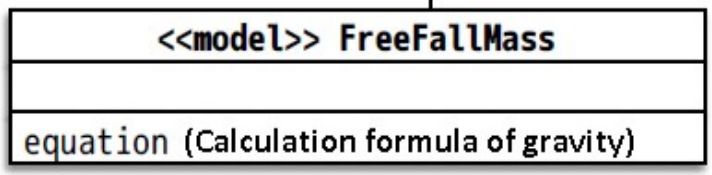
(1)



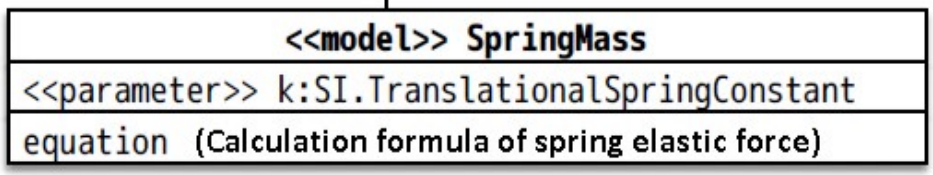
Partial class variable, equation, algorithm
Classes that are not complete

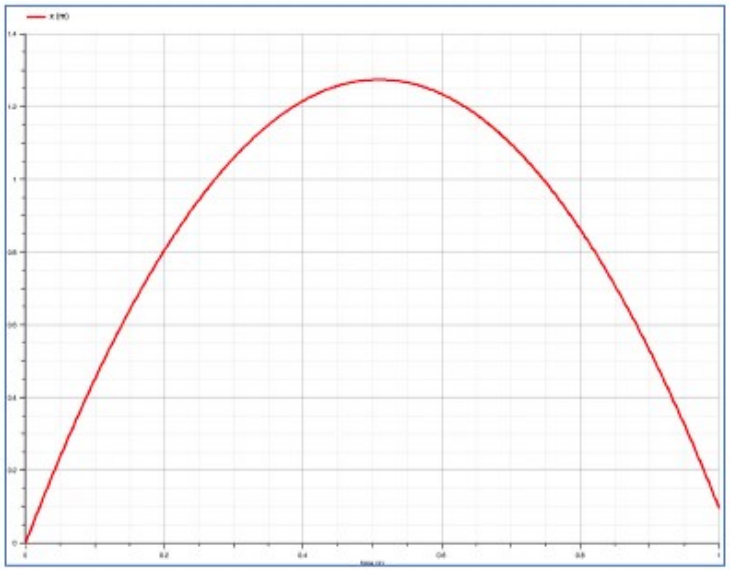
extends (Inherit, extend)
Create a new class by inheriting the existing class.

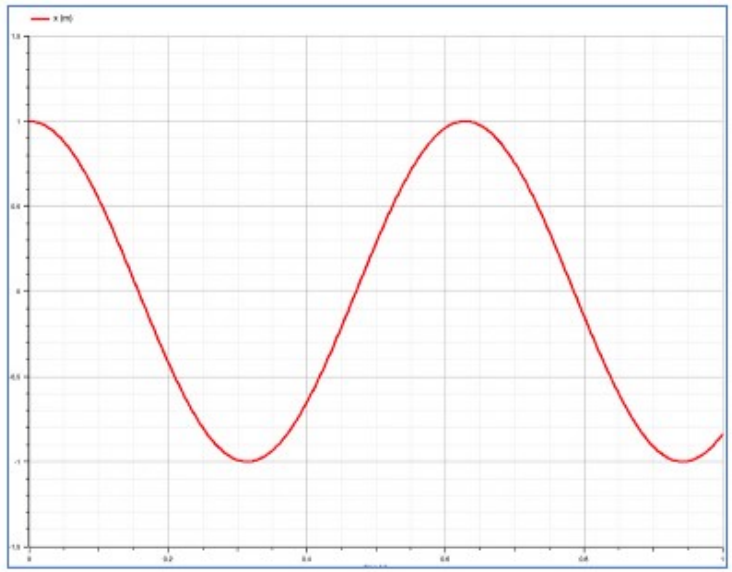
(2)



(3)







ClassExamle5

```
package ClassExample5
  import SI=Modelica.SIunits;
  constant SI.Acceleration g = -1 * Modelica.Constants.g_n;

  // partial model
  partial model PartialMass  (1)
    parameter SI.Mass m = 1.0;
    parameter SI.Velocity v0 = 5.0;
    parameter SI.Position x0 = 0.0;
    SI.Force f;
    SI.Position x(start = x0);
    SI.Velocity v(start = v0);
  equation
    v = der(x);
    f = m * der(v);
  end PartialMass;

  // model
  model FreeFallMass extends PartialMass;  (2)
  equation
   f = m * g;
  end FreeFallMass;

  model SpringMass extends PartialMass(x0 = 1.0, v0 = 0.0);  (3)
    parameter SI.TranslationalSpringConstant k = 100;
  equation
   f = -k * x;
  end SpringMass;

end ClassExample5;
```

Create an import statement alias to simplify class references.

Not enough equations for number of variables !!

3 variables

2 equations (mass equation of motion)

The model is completed by adding equations in the model of the inherited model !!!

Equation to calculate gravity as force

Equation to calculate spring elastic force as force

ClassExamle6

Create a local class

Implement the equation of motion

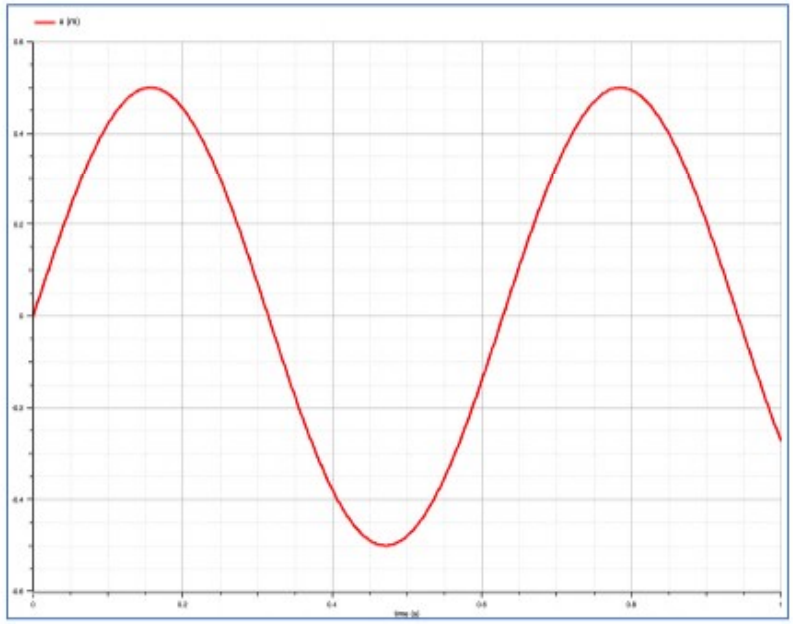
<<model>> MassC
<<parameter>> m: SI.Mass <<parameter>> v0: SI.Velocity <<parameter>> x0: SI.Position <<variable>> x: SI.Position <<variable>> v: SI.Velocity <<variable>> af: AppliedForce
model AppliedForce equation (Mass equation of motion)

Create a class inside a **local class**

By creating a local class, you can separate roles for each class.

Implement the formula for calculating the acting force

<<model>> AppliedForce
<<parameter>> k:SI.TranslationalSpringConstant <<input variable>> x: SI.Position <<output variable>> f: SI.Force
equation (Calculation formula of spring elastic force)



MassC calculation result

ClassExamle6

```
package ClassExample6 (1)
  import SI = Modelica.SIunits;
  constant SI.Acceleration g = -1 * Modelica.Constants.g_n;

  // model
  model MassC  (2)
    parameter SI.Mass m = 1.0;
    parameter SI.Velocity v0 = 0.0;
    parameter SI.Position x0 = 1.0;
    SI.Position x(start = x0);
    SI.Velocity v(start = v0);
    SI.Force f;
    AppliedForce af(k=100);

    // local model
    model AppliedForce  (3)
      parameter SI.TranslationalSpringConstant k = 100;
      input SI.Position x;
      output SI.Force f;
    algorithm
      f := -k * x;
    end AppliedForce;

  equation
    v = der(x);
    f = m * der(v);
    x = af.x;
    f = af.f;
  end MassC;

end ClassExample6;
```

Model representing motion of mass point

Local class (local model) object

Local model declaration

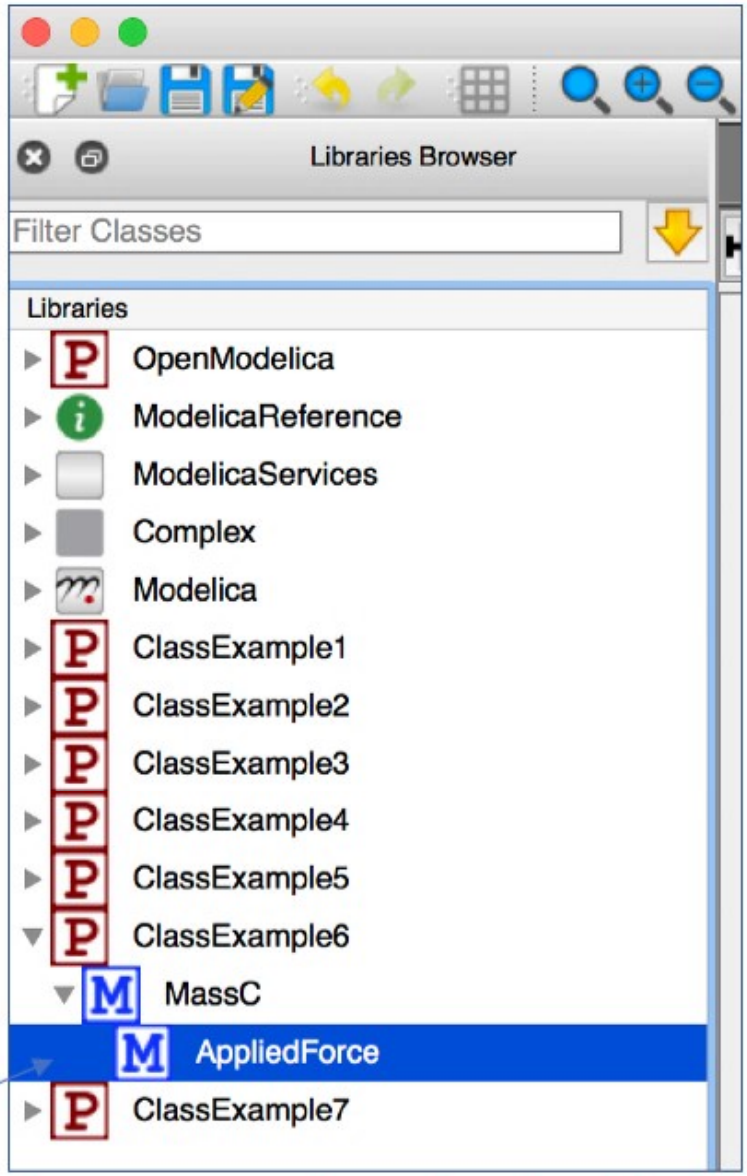
Local model to calculate the force acting on the mass point

Calculate the elastic force by Hooke's law

Mass equation of motion

Expression that represents the connection relationship with the local model

ClassExamle6



Local classes can be checked in the library browser.

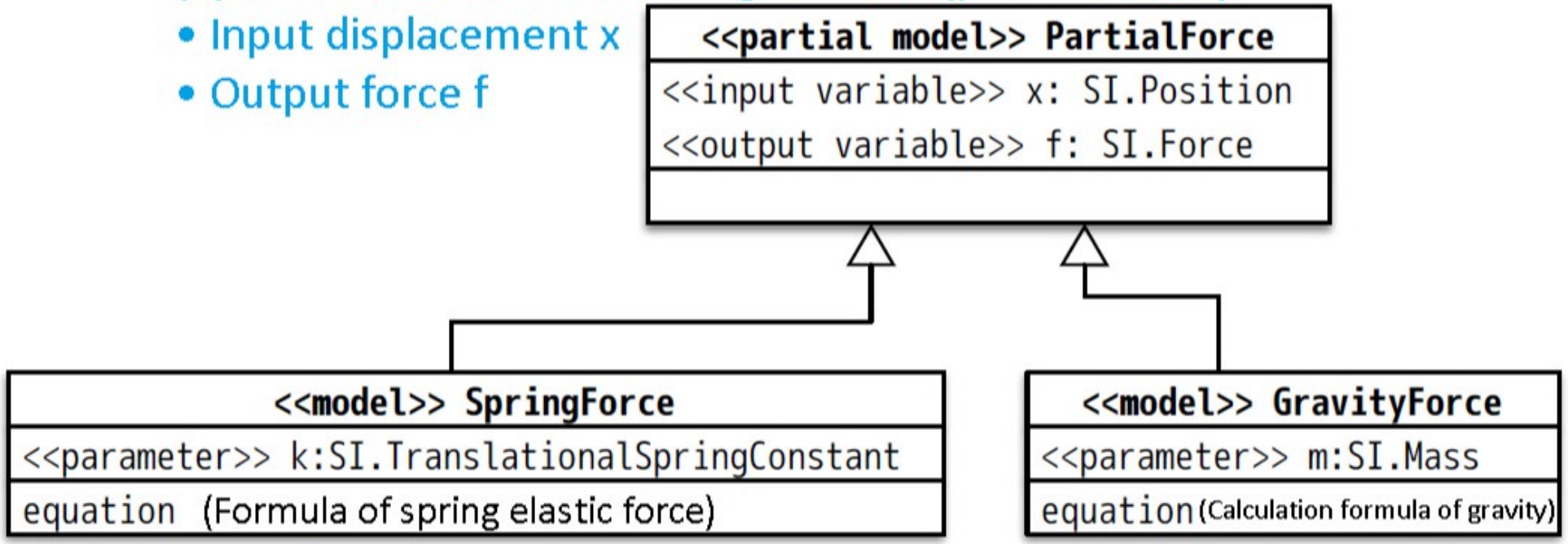
ClassExamle7

Create an replaceable local class

First, make several models of forces acting on the mass point.

(1) Base model of force acting on mass (partial model)

- Input displacement x
- Output force f



(2) Model of spring elastic force

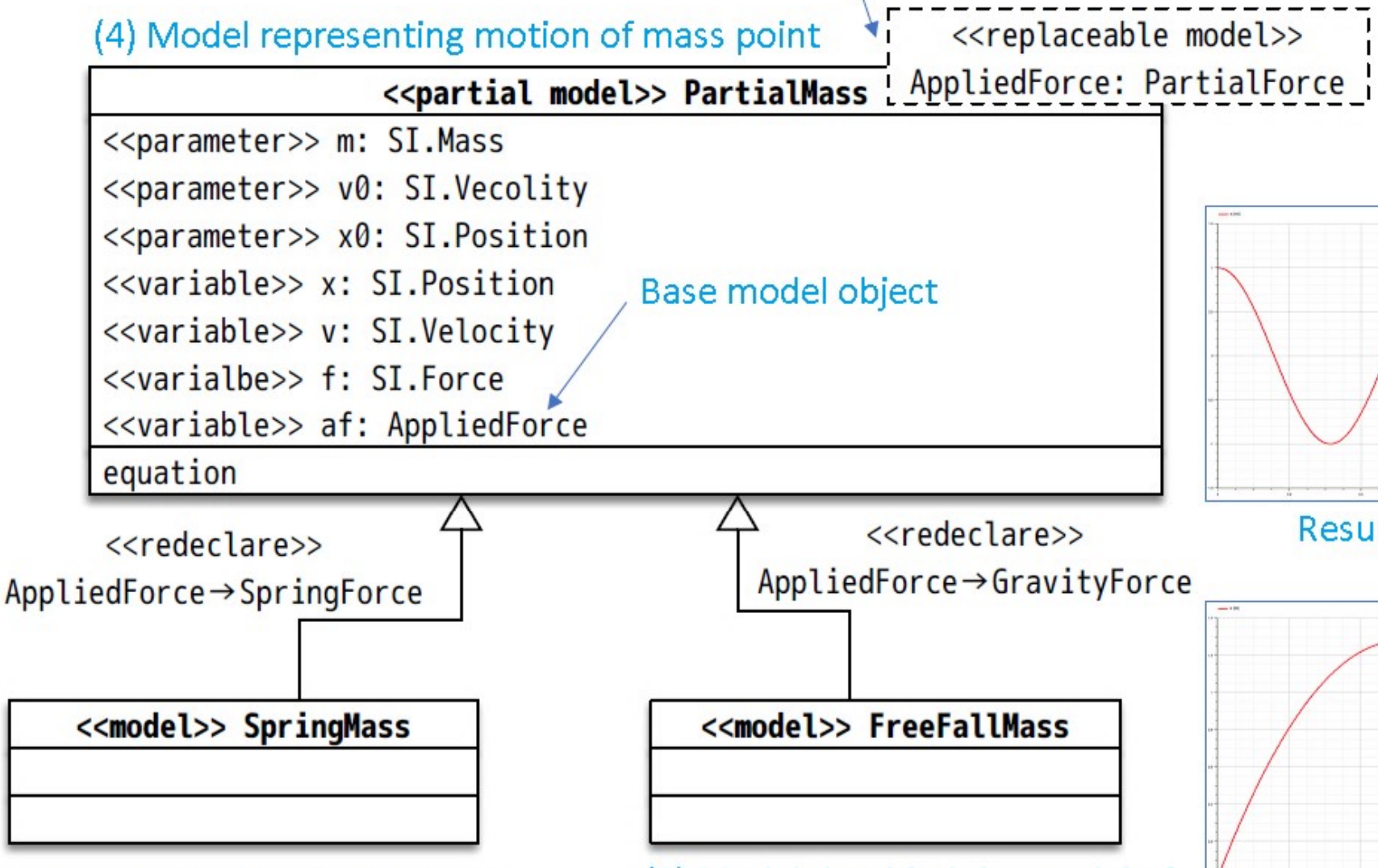
(3) Gravity model

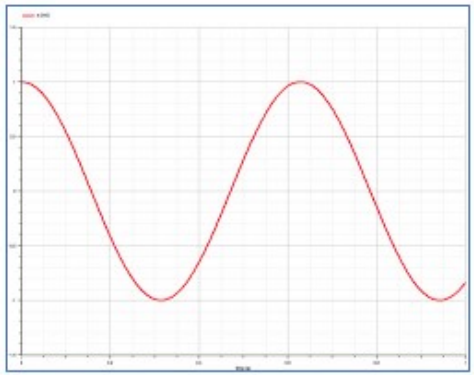
ClassExamle7

Create a class with a replaceable local class.

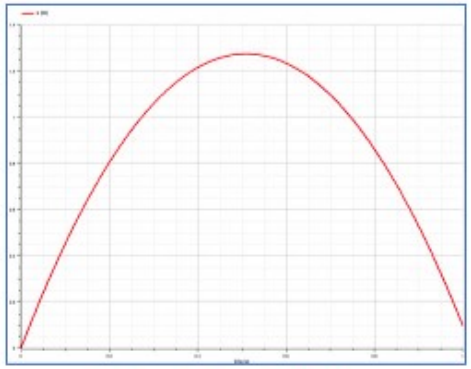
Replaceable local classes are written at the top right of the class like templates.

(4) Model representing motion of mass point





Result of (5)



Result of (6)

(5) Model that binds the model of spring elastic force

(6) Model that bind the model of gravity

ClassExamle7

```
package ClassExample7
  import SI = Modelica.SIunits;
  constant SI.Acceleration g = -1 * Modelica.Constants.g_n;

  // force models
  partial model PartialForce
    input SI.Position x;
    output SI.Force f;
  end PartialForce;

  model SpringForce extends PartialForce;
    parameter SI.TranslationalSpringConstant k = 1;
  algorithm
    f := -k * x;
  end SpringForce;

  model GravityForce extends PartialForce;
    parameter SI.Mass m = 1.0;
  algorithm
    f := m * g;
  end GravityForce;
```

(1) Base model of force acting on mass (partial model)

(2) Model of spring elastic force

(3) Gravity model

ClassExamle7

```
// mass models
 model PartialMass                                    (4) Model representing motion of mass point
   parameter SI.Mass m = 1.0;
   parameter SI.Velocity v0 = 5.0;
   parameter SI.Position x0 = 0.0;
   SI.Position x(start = x0);
   SI.Velocity v(start = v0);
   SI.Force f;
   AppliedForce af;
   replaceable model AppliedForce = PartialForce;
 equation
   v = der(x);
   f = m * der(v);
   x = af.x;
   f = af.f;
 end PartialMass;

 model SpringMass                                     (5) Mass connected to spring
 extends PartialMass(m=1.0,  x0=1.0, v0=0.0, redeclare model AppliedForce = SpringForce(k=100));
 end SpringMass;

 model FreeFallMass                                   (6) Free falling mass
 extends PartialMass(m = 1.0, v0=5.0, x0=0.0, redeclare model AppliedForce = GravityForce(m=1.0));
 end FreeFallMass;

end ClassExample7;
```

replaceable class Declares a local class whose class name can be replaced.

redeclare class classname
Redeclares a local class

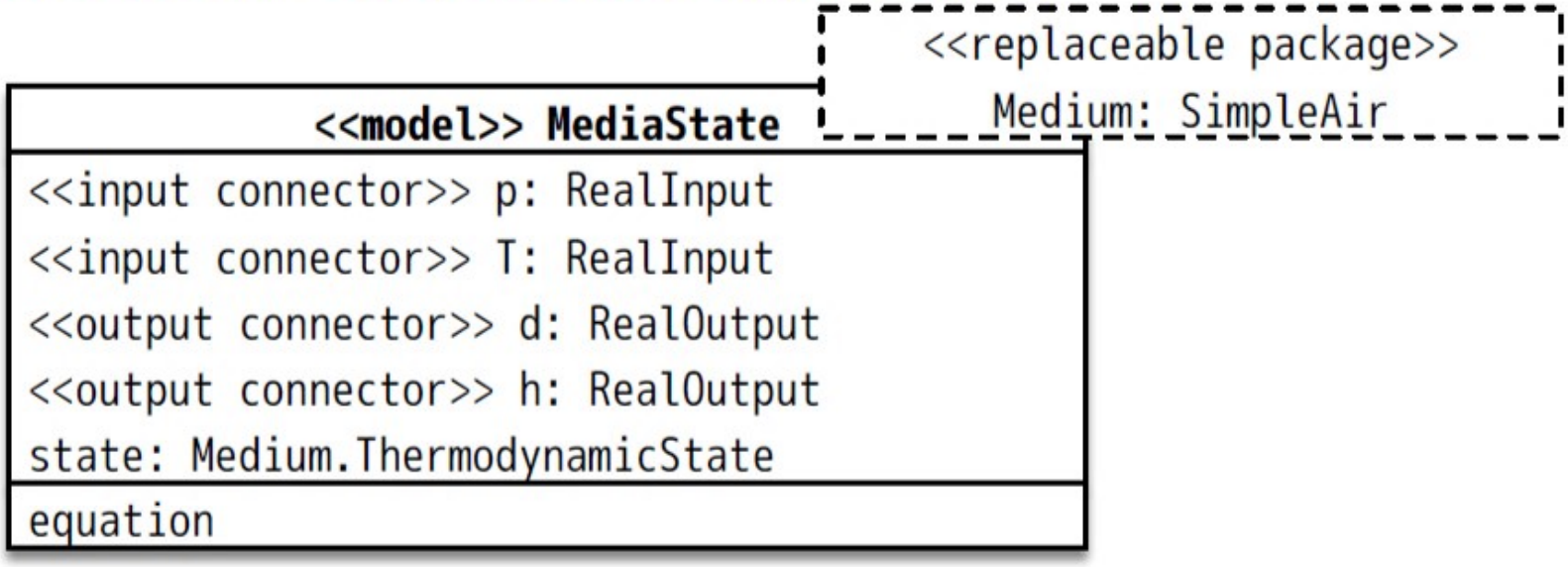
Bind the spring elasticity model.

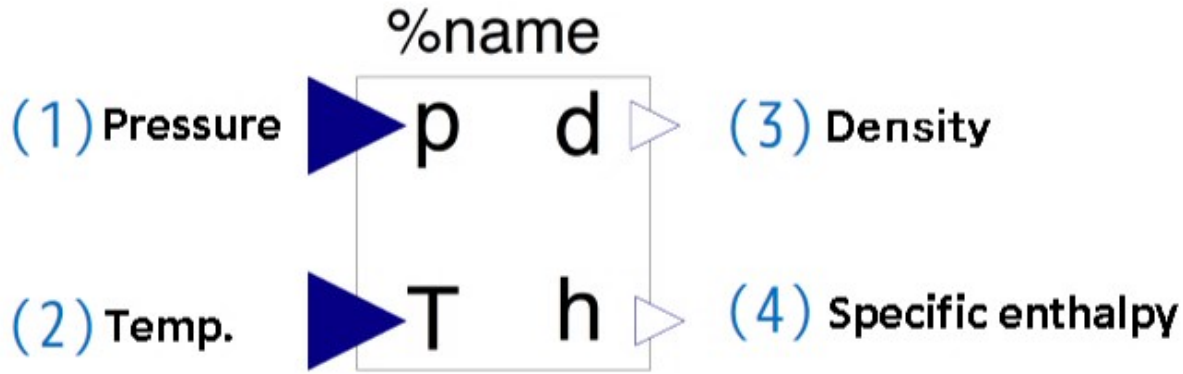
Bind the gravity model.

ClassExamle8

Examine the physical properties of air using Modelica.Media

Creates a MediaState model that return the density and specific enthalpy of air when provide temperature and pressure.





ClassExamle8

Declare Medium as an exchangeable local package and bind SimpleAir (physical model of air).

```
package ClassExample8
  import Modelica.Media;

  model MediaState
    replaceable package Medium = Media.Air.SimpleAir;
    Medium.ThermodynamicState state;
    Modelica.Blocks.Interfaces.RealInput p annotation( ...); (1)
    Modelica.Blocks.Interfaces.RealInput T annotation( ...); (2)
    Modelica.Blocks.Interfaces.RealOutput d annotation( ...); (3)
    Modelica.Blocks.Interfaces.RealOutput h annotation( ...); (4)
  equation
    state = Medium.setState_pT(p, T);
    d = Medium.density(state);
    h = Medium.specificEnthalpy(state);
    annotation( ...);
  end MediaState;
```

Record variable representing thermodynamic state

Input / output connector for pressure, temperature, density, specific enthalpy

The variable **state** representing the thermodynamic state is determined from the pressure **p** and the temperature **T**.

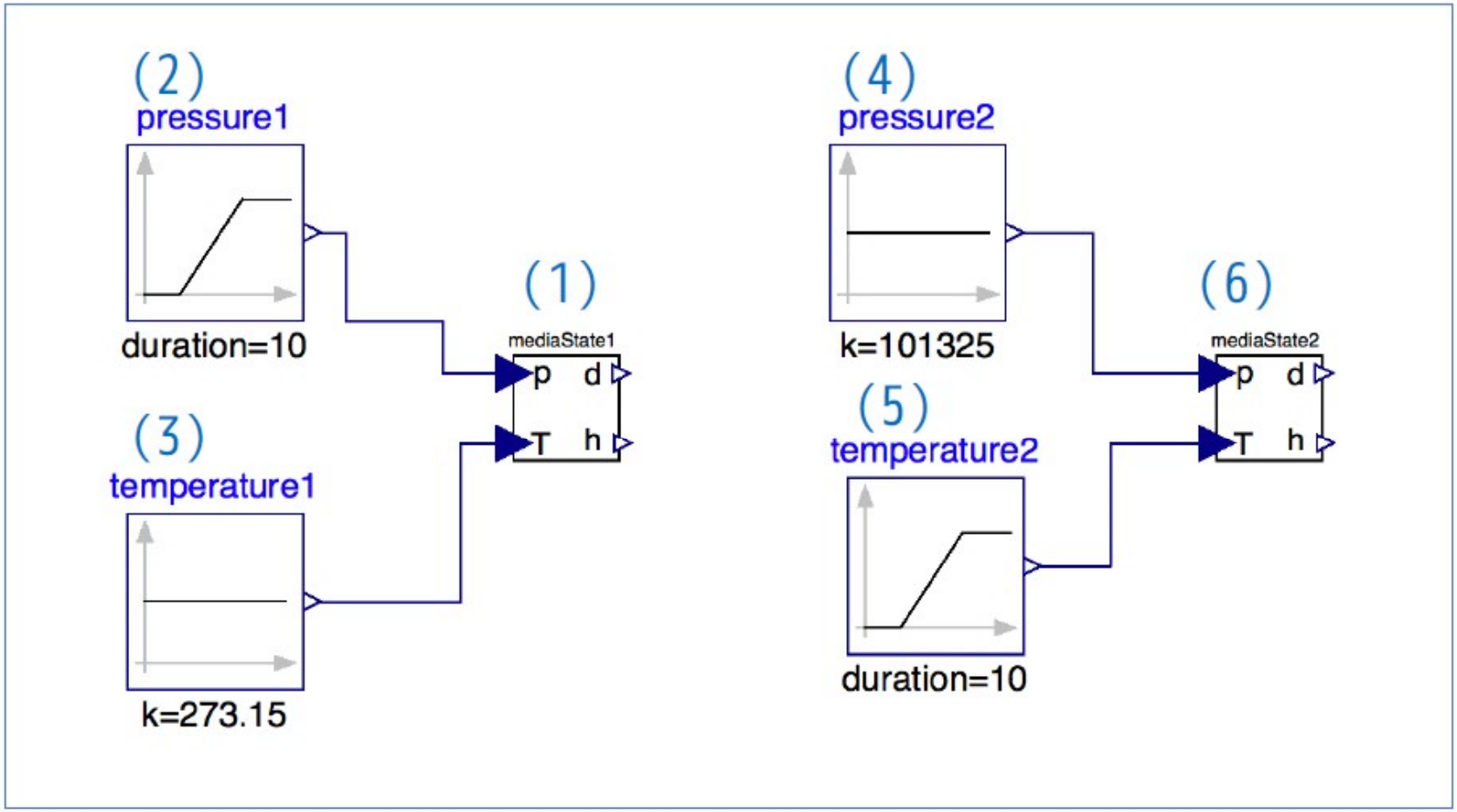
The density **d** and the specific enthalpy **h** are obtained from the variable **state**.

ClassExamle8

Test model MediaStateTest

Changing the pressure
while fixing the temperature

Changing pressure and
fixing temperature



ClassExamle8

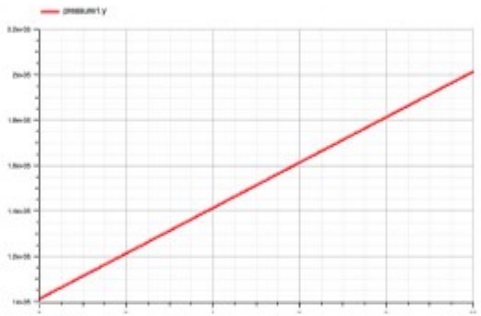
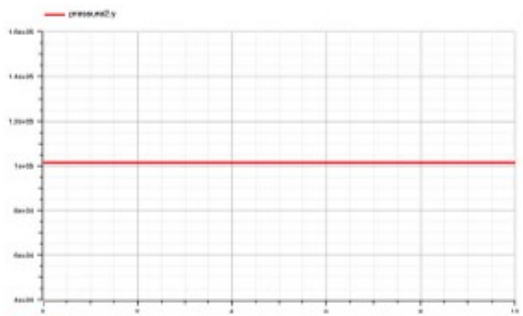
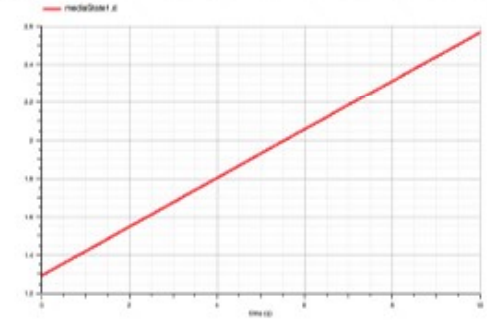
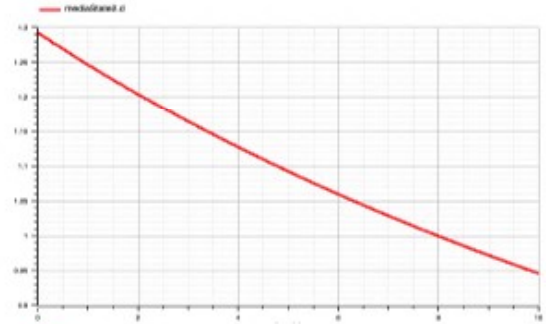
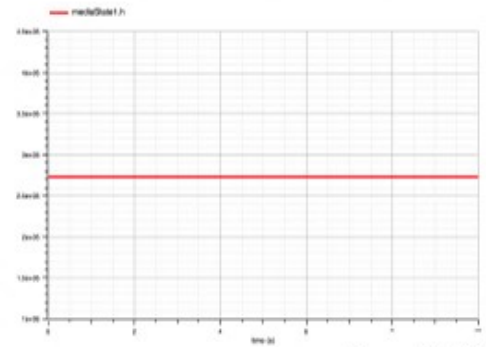
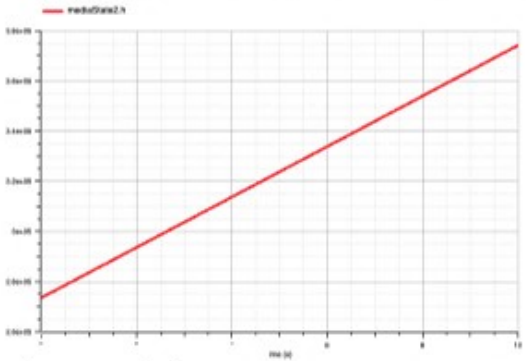
Change the physical model to DryAirNasa

```
model MediaStateTest
  replaceable package Medium = Media.Air.DryAirNasa;
  MediaState mediaState1(redeclare package Medium = Medium) annotation( ...);              (1)
  Modelica.Blocks.Sources.Ramp pressure1(duration = 10, height = 100000, offset = 101325)  (2)
    annotation( ...);
  Modelica.Blocks.Sources.Constant temperature1(k = 273.15)  annotation( ...);             (3)
  Modelica.Blocks.Sources.Constant pressure2(k = 101325) annotation( ...);                 (4)
  Modelica.Blocks.Sources.Ramp temperature2(duration = 10, height = 100, offset = 273.15)  (5)
   annotation( ...);
  MediaState mediaState2(redeclare package Medium = Medium) annotation( ...);              (6)
equation
  connect(temperature2.y, mediaState2.T) annotation( ...);
  connect(pressure2.y, mediaState2.p) annotation( ...);
  connect(pressure1.y, mediaState1.p) annotation( ...);
  connect(temperature1.y, mediaState1.T) annotation( ...);
end MediaStateTest;
```

With such attitude, you can investigate the properties of various types of fluids (pure substances).

ClassExamle8

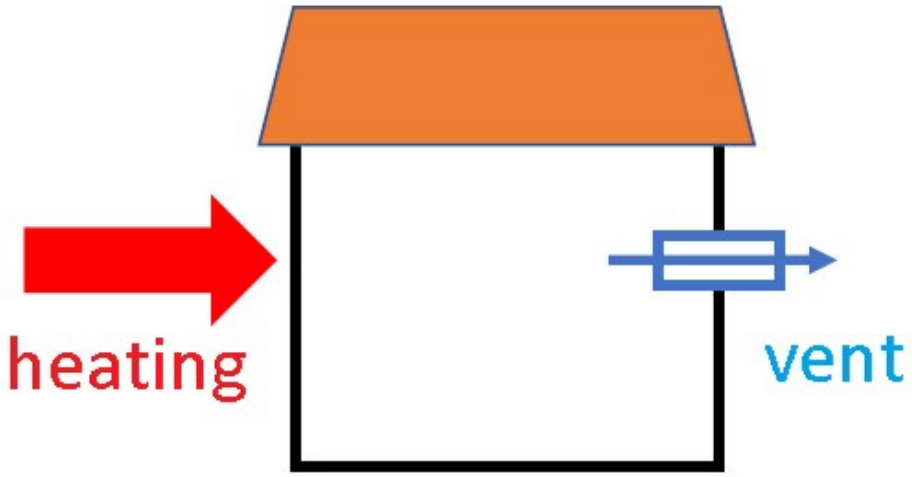
Test model
calculation results

	mediaState1	mediaState2
p		
T		
d		
h		

ClassExamle9

Examine room temperature changes using Modelica.Media

- Heat the room at a constant pressure.
- Rooms have vents that allow expanded air to escape.



ClassExamle9

First, create a simple model.

```
package ClassExample9
  import Modelica.Media;
  import SI = Modelica.SIunits;

  model RoomA
    package Medium = Media.Air.DryAirNasa;
    parameter SI.Temperature T_amb = 293.15;
    parameter SI.Pressure p_amb = 101325;
    parameter SI.Volume V = 22.0;
    parameter SI.HeatFlowRate Q_flow = 100;

    Medium.BaseProperties medium;
    SI.MassFlowRate m_flow(start = 0.0);
    SI.Mass M;
    SI.Energy U;
  equation
    M = medium.d * V;
    U = medium.u * M;
    der(M) = m_flow;
    der(U) = Q_flow + medium.h * m_flow;
    medium.p = p_amb;
  initial equation
    medium.T = T_amb;
  end RoomA;
```

Declare DryAirNasa (physical model of air) as a local package Medium.

Atmospheric condition

Room volume (about 4.5 tatami mats)

Heat flow 100 W

Initial temperature

Equation

$$M = \rho V$$

$$U = Mu$$

Air mass and internal energy throughout the room

$$\frac{dM}{dt} = m\_flow$$

Mass conservation law and energy conservation law

$$\frac{dU}{dt} = Q\_flow + h \cdot m\_flow$$

$$p = p_{amb}$$

Pressure is fixed at ambient pressure

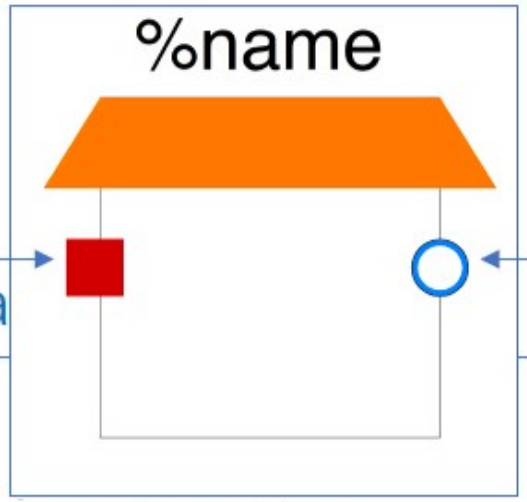
ClassExamle9

Componentize using connectors

```
model RoomB
  replaceable package Medium = Media.Air.DryAirNasa;
  parameter SI.Volume V = 22.0;
  Modelica.Fluid.Interfaces.FluidPort_b port_b(redeclare package Medium = Medium) (1)
    annotation( ...);
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port_a annotation( ...);    (2)
  Medium.BaseProperties medium;
  SI.Mass M;
  SI.Energy U;
equation
  M = medium.d * V;
  U = medium.u * M;
  der(M) = port_b.m_flow;
  der(U) = port_a.Q_flow + actualStream(port_b.h_outflow) * port_b.m_flow;
  port_b.p = medium.p;
  port_b.h_outflow = medium.h;
  port_a.T = medium.T;
initial equation
  medium.T = 293.15;
  annotation( ...);
end RoomB;
```

Set the fluid flowing through port_b.

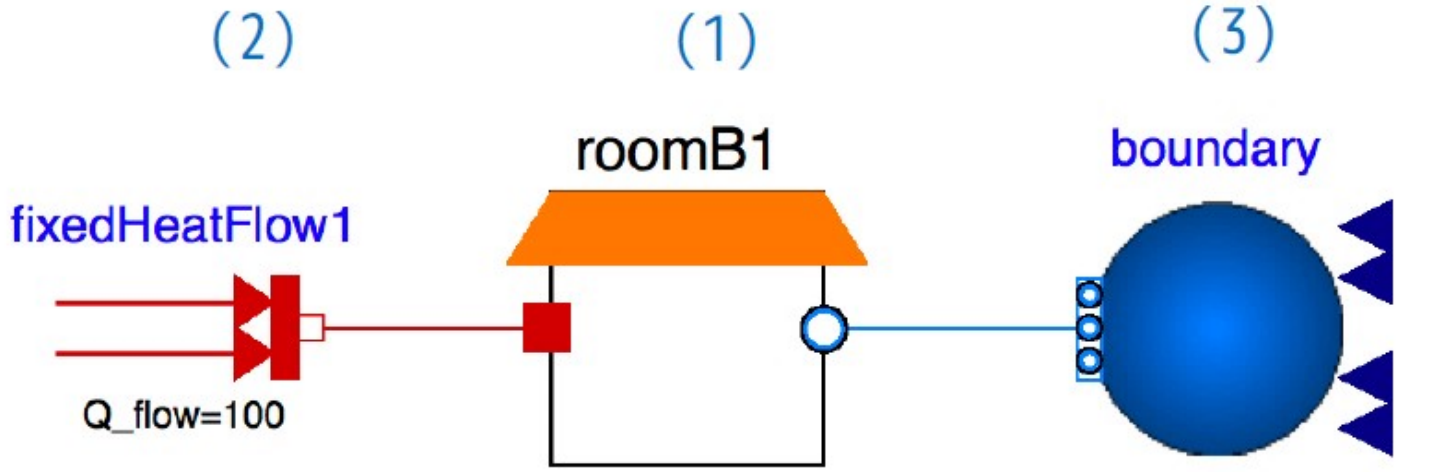
(2)
port_a:
HeatPort_a



(1)
port_b:
FluidPort_b

ClassExamle9

Create a test model



ClassExamle9

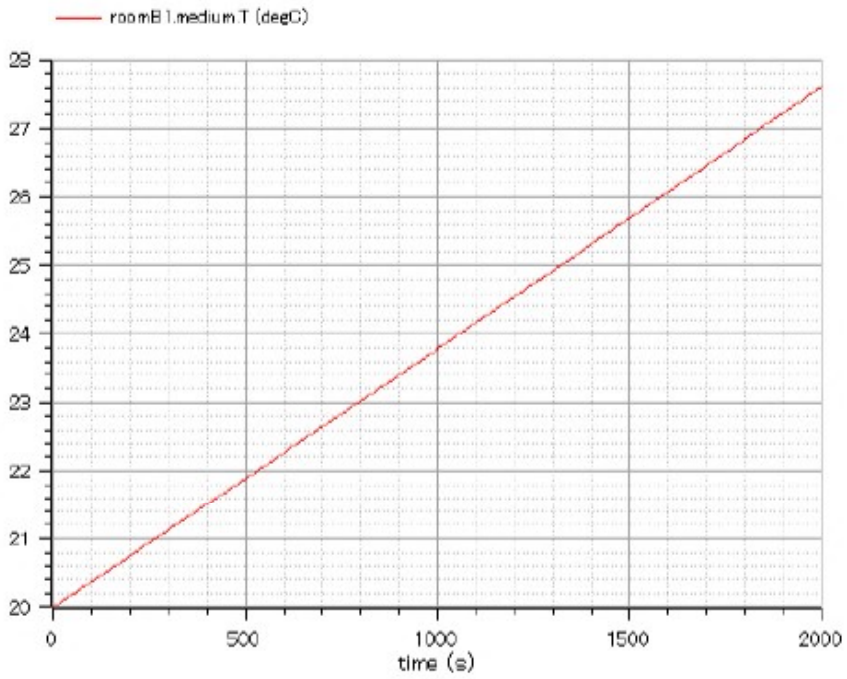
RoomTest source code

Set the fluid flowing through the component.

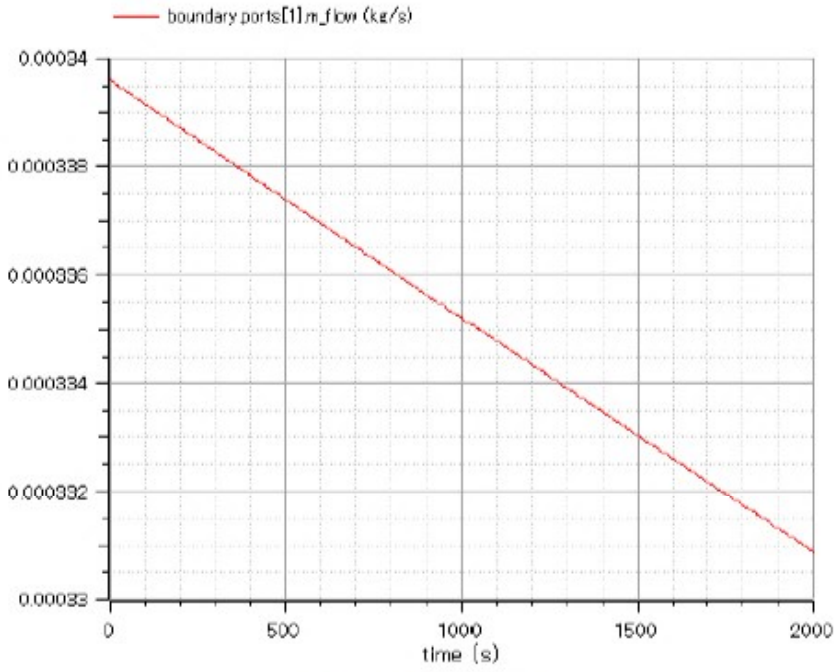
```
model RoomBTest
  replaceable package Medium = Media.Air.DryAirNasa;
  ClassExample9.RoomB roomB1(redeclare package Medium = Medium)                    (1)
    annotation( ...);
  Modelica.Thermal.HeatTransfer.Sources.FixedHeatFlow fixedHeatFlow1(Q_flow = 100)  (2)
    annotation( ...);
  Modelica.Fluid.Sources.Boundary_pT boundary(redeclare package Medium = Medium,    (3)
    T = 293.15, nPorts = 1, p = 101325)  annotation( ...);
equation
  connect(roomB1.port_b, boundary.ports[1]) annotation( ...);
  connect(fixedHeatFlow1.port, roomB1.port_a) annotation( ...);
end RoomBTest;

  annotation( ...);
end ClassExample9;
```

ClassExamle9

Test model simulation results



Changes in room temperature



Vent flow rate

Conclusion

- The features of eight classes of Modelica were shown.
- Described the basic use of classes.

Modelica classes

① package
② type
③ class
④ record
⑤ model
⑥ function
⑦ connector
⑧ block

Basic use of classes

- Connection by connector
- Reference by alias (import statement)
- Inherits and partial classes
- Local class
- Exchangeable local classes

(replaceable, redeclare)

- The purpose of this document is introducing Media and Fluid Libraries in the Modelica Standard Library (MSL). This document uses libraries, software, figures, and documents included in MSL and those modifications. Licenses and copyrights of those are written in next page.
- Copyright and License of this document are written in the last page.

Modelica Standard Library License

https://github.com/modelica/ModelicaStandardLibrary/blob/master/LICENSE

BSD 3-Clause License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.