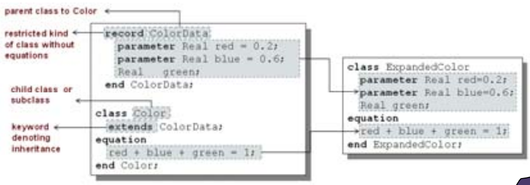
Modelica Language Concepts and Textual Modeling

Classes and Inheritance

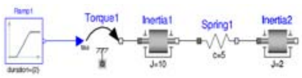
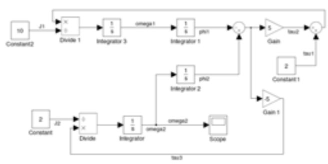


Typed Declarative Equation-based Textual Language

Hybrid Modeling

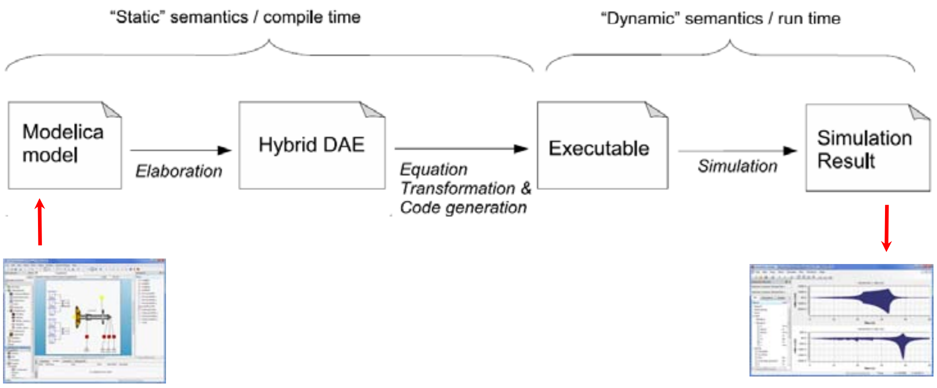
Acausal Modeling

The order of computations is not decided at modeling time

	Acausal	Causal
Visual Component Level		

Equation Level	A resistor *equation*:
	`R*i = v;`
		Causal possibilities:
		`i := v/R;`
		`v := R*i;`
		`R := v/i;`

Typical Simulation Process

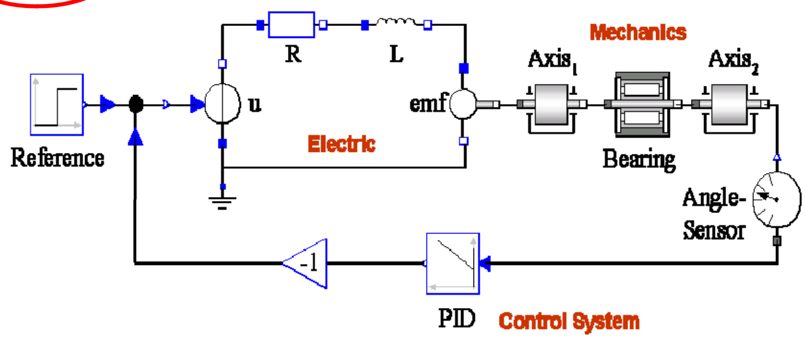


What is Special about Modelica?

- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

What is Special about Modelica?

Multi-Domain Modeling



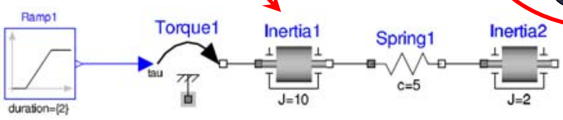
What is Special about Modelica?

Multi-Domain Modeling

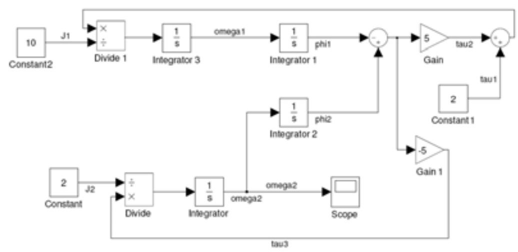
Keeps the physical structure

Visual Acausal Hierarchical Component Modeling

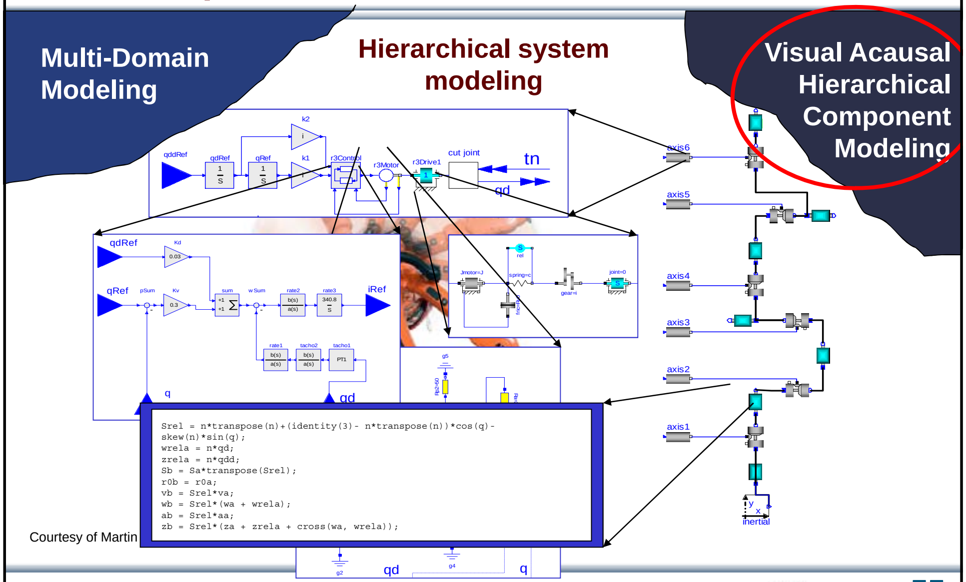
Acausal model (Modelica)



Causal block-based model (Simulink)



What is Special about Modelica?



What is Special about Modelica?

Multi-Domain Modeling

A textual *class-based* language
OO primary used for as a structuring concept

Visual Acausal Hierarchical Component Modeling

Behaviour described declaratively using
- Differential algebraic equations (DAE) (continuous-time)
- Event triggers (discrete-time)

Variable declarations

```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x";
  Real y(start = 1)  "y coordinate";
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
```

Differential equations

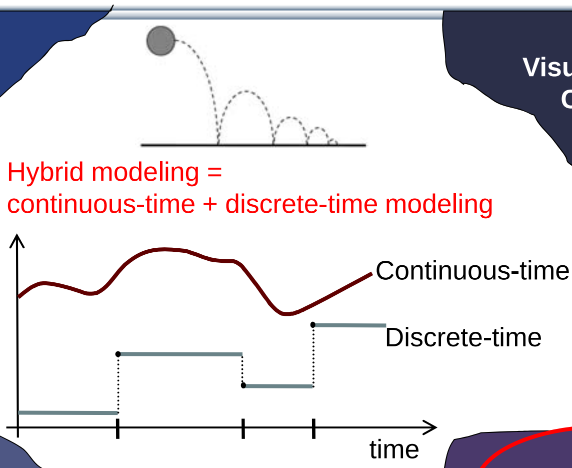
Typed Declarative Equation-based Textual Language

What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Component Modeling

Hybrid modeling =
continuous-time + discrete-time modeling



Typed Declarative Equation-based Textual Language

Hybrid Modeling

Modelica Classes and Inheritance

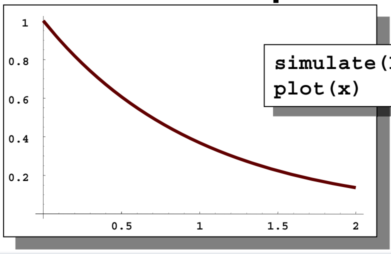
Simplest Model – Hello World!

A Modelica "Hello World" model

Equation: x' = - x
Initial condition: x(0) = 1

```
class HelloWorld "A simple equation"
  Real x(start=1);
equation
  der(x)= -x;
end HelloWorld;
```

Simulation in OpenModelica environment




```
simulate(HelloWorld, stopTime = 2)
plot(x)
```

  MODELICA pelab

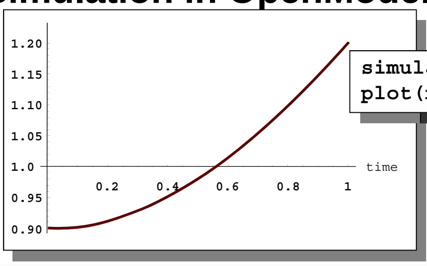
Model Including Algebraic Equations

Include algebraic equation

Algebraic equations contain no derivatives

```
class DAEexample
  Real x(start=0.9);
  Real y;
equation
  der(y)+(1+0.5*sin(y))*der(x)
    = sin(time);
  x - y = exp(-0.9*x)*cos(y);
end DAEexample;
```

Simulation in OpenModelica environment



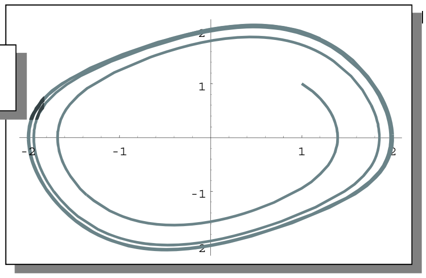

```
simulate(DAEexample, stopTime = 1)
plot(x)
```

  MODELICA pelab

Example class: Van der Pol Oscillator

```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x"; // x starts at 1
  Real y(start = 1)  "y coordinate";             // y starts at 1
  parameter Real lambda = 0.3;
equation
  der(x) = y;                          // This is the 1st diff equation //
  der(y) = -x + lambda*(1 - x*x)*y;   /* This is the 2nd diff equation */
end VanDerPol;
```

```
simulate(VanDerPol,stopTime = 25)
plotParametric(x,y)
```



Exercises – Simple Textual Modeling

- Start OMNotebook
  - Start->Programs->OpenModelica->OMNotebook
  - Open File: Exercise01-classes-simple-textual.onb
- Open Exercise01-classes-simple-textual.pdf

Exercises 2.1 and 2.2

- Open the **Exercise01-classes-simple-textual.onb** found in the Tutorial directory.
- Locate the VanDerPol model in DrModelica (link from Section 2.1), using OMNotebook!
- **Exercise 2.1**: Simulate and plot VanDerPol. Do a slight change in the model, re-simulate and re-plot.
- **Exercise 2.2**. Simulate and plot the HelloWorld example. Do a slight change in the model, re-simulate and re-plot. Try command-completion, val( ), etc.

```
class HelloWorld "A simple equation"
  Real x(start=1);
equation
  der(x)= -x;
end HelloWorld;
```

```
simulate(HelloWorld, stopTime = 2)
plot(x)
```


Variables and Constants

Built-in primitive data types

Boolean	**true** or **false**
Integer	Integer value, e.g. **42** or **–3**
Real	Floating point value, e.g. **2.4e-6**
String	String, e.g. **“Hello world”**
Enumeration	Enumeration literal e.g. **ShirtSize.Medium**

Variables and Constants cont'

- Names indicate meaning of constant
- Easier to maintain code
- Parameters are constant during simulation
- Two types of constants in Modelica
  - **constant**
  - **parameter**

```
constant  Real    PI=3.141592653589793;
constant  String  redcolor = "red";
constant  Integer one = 1;
parameter Real    mass = 22.5;
```

 Peter Fritzson

Comments in Modelica

1) Declaration comments, e.g. `Real x "state variable";`

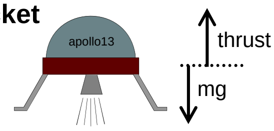
```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x"; // x starts at 1
  Real y(start = 1)  "y coordinate";             // y starts at 1
  parameter Real lambda = 0.3;
equation
  der(x) = y;                          // This is the 1st diff equation //
  der(y) = -x + lambda*(1 - x*x)*y;   /* This is the 2nd diff equation */
end VanDerPol;
```

2) Source code comments, disregarded by compiler
   2a) C style, e.g. `/* This is a C style comment */`
   2b) C++ style, e.g. `// Comment to the end of the line…`

 Peter Fritzson

A Simple Rocket Model

Rocket



$$acceleration = \frac{thrust - mass \cdot gravity}{mass}$$

$$mass' = -massLossRate \cdot \text{abs}(thrust)$$

$$altitude' = velocity$$

$$velocity' = acceleration$$

```
class Rocket "rocket class"
  parameter String name;
  Real mass(start=1038.358);
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;  // Thrust force on rocket
  Real gravity; // Gravity forcefield
  parameter Real massLossRate=0.000277;
equation
  (thrust-mass*gravity)/mass = acceleration;
  der(mass)   = -massLossRate * abs(thrust);
  der(altitude) = velocity;
  der(velocity) = acceleration;
end Rocket;
```

- new model → `class Rocket`
- declaration comment → `"rocket class"`
- parameters (changeable before the simulation) → `parameter String`
- floating point type → `Real`
- start value → `start= -2003`
- name + default value → `massLossRate=0.000277`
- mathematical equation (acausal) → `(thrust-mass*gravity)/mass = acceleration;`
- differentiation with regards to time → `der(velocity)`

Celestial Body Class

A class declaration creates a *type name* in Modelica

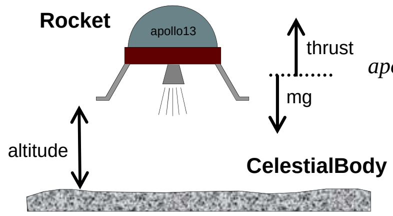
```
class CelestialBody
  constant  Real   g = 6.672e-11;
  parameter Real   radius;
  parameter String name;
  parameter Real   mass;
end CelestialBody;
```

An *instance* of the class can be declared by *prefixing* the type name to a variable name

```
...
CelestialBody moon;
...
```

The declaration states that **`moon`** is a variable containing an object of type **`CelestialBody`**

Moon Landing



$$apollo.gravity = \frac{moon.g \cdot moon.mass}{(apollo.altitude + moon.radius)^2}$$

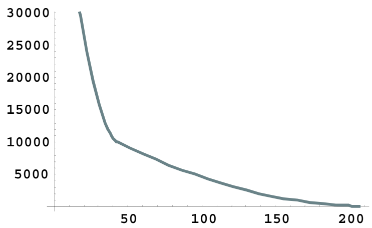
only access inside the class ← `protected`

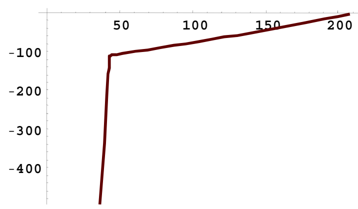
access by dot notation outside the class ← `public`

```
class MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
protected
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
public
  Rocket        apollo(name="apollo13");
  CelestialBody   moon(name="moon",mass=7.382e22,radius=1.738e6);
equation
  apollo.thrust = if (time < thrustDecreaseTime) then force1
                  else if (time < thrustEndTime) then force2
                  else 0;
  apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end MoonLanding;
```


Simulation of Moon Landing

```
simulate(MoonLanding, stopTime=230)
plot(apollo.altitude, xrange={0,208})
plot(apollo.velocity, xrange={0,208})
```





It starts at an altitude of 59404 (not shown in the diagram) at time zero, gradually reducing it until touchdown at the lunar surface when the altitude is zero

The rocket initially has a high negative velocity when approaching the lunar surface. This is reduced to zero at touchdown, giving a smooth landing

Restricted Class Keywords

- The `class` keyword can be replaced by other keywords, e.g.: `model`, `record`, `block`, `connector`, `function`, ...
- Classes declared with such keywords have restrictions
- Restrictions apply to the contents of restricted classes

- Example: A `model` is a class that cannot be used as a connector class
- Example: A `record` is a class that only contains data, with no equations
- Example: A `block` is a class with fixed input-output causality

```
model CelestialBody
  constant  Real    g = 6.672e-11;
  parameter Real    radius;
  parameter String  name;
  parameter Real    mass;
end CelestialBody;
```


Modelica Functions

- Modelica Functions can be viewed as a special kind of restricted class with some extensions
- A function can be called with arguments, and is instantiated dynamically when called
- More on functions and algorithms later in Lecture 4

```
function sum
  input  Real arg1;
  input  Real arg2;
  output Real result;
algorithm
  result := arg1+arg2;
end sum;
```


Inheritance

parent class to Color

restricted kind of class without equations

```
record ColorData
  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real   green;
end ColorData;
```

child class or subclass

keyword denoting inheritance

```
class Color
  extends ColorData;
equation
  red + blue + green = 1;
end Color;
```

```
class ExpandedColor
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
equation
  red + blue + green = 1;
end ExpandedColor;
```

Data and behavior: field declarations, equations, and certain other contents are copied into the subclass

  MODELICA pelab

Inheriting definitions

```
record ColorData
  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real   green;
end ColorData;

class ErrorColor
  extends ColorData;
  parameter Real blue = 0.6;
  parameter Real red = 0.3;
equation
  red + blue + green = 1;
end ErrorColor;
```

Legal! Identical to the inherited field blue

Inheriting multiple identical definitions results in only one definition

Illegal! Same name, but different value

Inheriting multiple different definitions of the same item is an error

  MODELICA pelab

Inheritance of Equations

```
class Color
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
equation
  red + blue + green = 1;
end Color;
```

```
class Color2   // OK!
  extends Color;
equation
  red + blue + green = 1;
end Color2;
```

`Color` is identical to `Color2`

Same equation twice leaves one copy when inheriting

```
class Color3   // Error!
   extends Color;
equation
  red + blue + green = 1.0;
  // also inherited: red + blue + green = 1;
end Color3;
```

`Color3` is overdetermined

Different equations means two equations!

Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color

```
class Color
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
equation
  red + blue + green = 1;
end Color;
```

```
class Point
  Real x;
  Real y,z;
end Point;
```

```
class ColoredPoint
  extends Point;
  extends Color;
end ColoredPoint;
```

multiple inheritance

Equivalent to

```
class ColoredPointWithoutInheritance
  Real x;
  Real y, z;
  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real green;
equation
  red + blue + green = 1;
end ColoredPointWithoutInheritance;
```


Multiple Inheritance cont'

Only one copy of multiply inherited class `Point` is kept

```
class Point
  Real x;
  Real y;
end Point;
```

```
class VerticalLine
  extends Point;
  Real vlength;
end VerticalLine;
```

Diamond Inheritance

```
class HorizontalLine
  extends Point;
  Real hlength;
end HorizontalLine;
```

```
class Rectangle
  extends VerticalLine;
  extends HorizontalLine;
end Rectangle;
```


Simple Class Definition – Shorthand Case of Inheritance

- Example:

```
class SameColor = Color;
```

Equivalent to:

inheritance ←

```
class SameColor
  extends Color;
end SameColor;
```

- Often used for introducing new names of types:

```
type Resistor = Real;
```

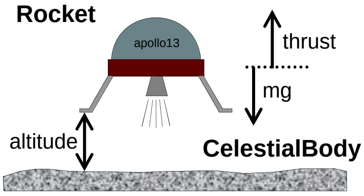
```
connector MyPin = Pin;
```


Inheritance Through Modification

- Modification is a concise way of combining inheritance with declaration of classes or instances
- A modifier modifies a declaration equation in the inherited class
- Example: The class `Real` is inherited, modified with a different `start` value equation, and instantiated as an `altitude` variable:

```
...
 Real altitude(start= 59404);
...
```


The Moon Landing Example Using Inheritance




```
model Rocket "generic rocket class"
  extends Body;
  parameter Real massLossRate=0.000277;
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;
  Real gravity;
equation
  thrust-mass*gravity= mass*acceleration;
  der(mass)= -massLossRate*abs(thrust);
  der(altitude)= velocity;
  der(velocity)= acceleration;
end Rocket;
```

```
model Body "generic body"
  Real   mass;
  String name;
end Body;
```

```
model CelestialBody
  extends Body;
  constant  Real g = 6.672e-11;
  parameter Real radius;
end CelestialBody;
```


The Moon Landing
Example using Inheritance cont'

inherited parameters

```
model MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
  Rocket          apollo(name="apollo13", mass(start=1038.358) );
  CelestialBody   moon(mass=7.382e22,radius=1.738e6,name="moon");
equation
  apollo.thrust = if (time<thrustDecreaseTime) then force1
                  else if (time<thrustEndTime) then force2
                  else 0;
  apollo.gravity =moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;
```


Inheritance of Protected Elements

If an `extends`-clause is preceded by the `protected` keyword, all inherited elements from the superclass become protected elements of the subclass

```
class Point
  Real x;
  Real y,z;
end Point;
```

```
class Color
  Real red;
  Real blue;
  Real green;
equation
  red + blue + green = 1;
end Color;
```

```
class ColoredPoint
 protected
  extends Color;
 public
  extends Point;
end ColoredPoint;
```

Equivalent to

```
class ColoredPointWithoutInheritance
  Real x;
  Real y,z;
  protected Real red;
  protected Real blue;
  protected Real green;
equation
  red + blue + green = 1;
end ColoredPointWithoutInheritance;
```

The inherited fields from `Point` keep their protection status since that `extends`-clause is preceded by `public`

A protected element cannot be accessed via dot notation!

Advanced Topic

- Class parameterization

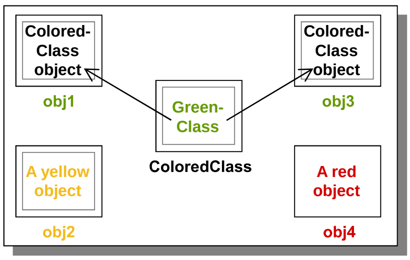
Generic Classes with Type Parameters

Formal class parameters are replaceable variable or type declarations within the class (usually) marked with the prefix `replaceable`

```
class C
  replaceable class ColoredClass = GreenClass;
  ColoredClass            obj1(p1=5);
  replaceable YellowClass obj2;
  ColoredClass            obj3;
  RedClass                obj4;
equation
end C;
```

Actual arguments to classes are modifiers, which when containing whole variable declarations or types are preceded by the prefix `redeclare`

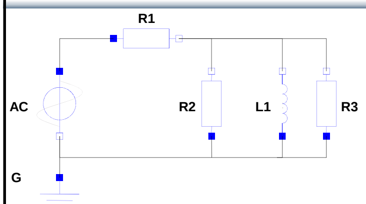
```
class C2 =
  C(redeclare class ColoredClass = BlueClass);
```



Equivalent to

```
class C2
  BlueClass   obj1(p1=5);
  YellowClass obj2;
  BlueClass   obj3;
  RedClass    obj4;
equation
end C2;
```


Class Parameterization when Class Parameters are Components



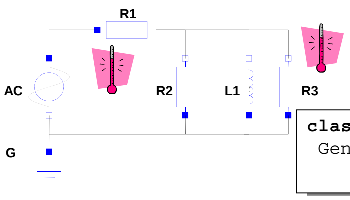
The class `ElectricalCircuit` has been converted into a parameterized generic class `GenericElectricalCircuit` with three formal class parameters `R1`, `R2`, `R3`, marked by the keyword `replaceable`

```
class ElectricalCircuit
  Resistor R1(R=100);
  Resistor R2(R=200);
  Resistor R3(R=300);
  Inductor L1;
  SineVoltage AC;
  Groung G;
equation
  connect(R1.n,R2.n);
  connect(R1.n,L1.n);
  connect(R1.n,R3.n);
  connect(R1.p,AC.p);
  .....
end ElectricalCircuit;
```

Class parameterization

```
class GenericElectricalCircuit
  replaceable Resistor R1(R=100);
  replaceable Resistor R2(R=200);
  replaceable Resistor R3(R=300);
  Inductor L1;
  SineVoltage AC;
  Groung G;
equation
  connect(R1.n,R2.n);
  connect(R1.n,L1.n);
  connect(R1.n,R3.n);
  connect(R1.p,AC.p);
  .....
end GenericElectricalCircuit;
```


Class Parameterization when Class Parameters are Components - cont'



A more specialized class `TemperatureElectricalCircuit` is created by changing the types of `R1`, `R3`, to `TempResistor`

```
class TemperatureElectricalCircuit =
  GenericElectricalCircuit (redeclare TempResistor R1
                            redeclare TempResistor R3);
```

```
class TemperatureElectricalCircuit
  parameter Real Temp=20;
  extends GenericElectricalCircuit(
    redeclare TempResistor R1(RT=0.1, Temp=Temp),
    redeclare TempResistor R3(R=300));
end TemperatureElectricalCircuit
```

We add a temperature variable `Temp` for the temperature of the resistor circuit and modifiers for `R1` and `R3` which are now `TempResistors`.

equivalent to

```
class ExpandedTemperatureElectricalCircuit
  parameter Real Temp;
  TempResistor R1(R=200, RT=0.1, Temp=Temp),
  replaceable Resistor R2;
  TempResistor R3(R=300);
equation
  ....
end ExpandedTemperatureElectricalCircuit
```


Exercises 1 Simple Textual Continued

- Continue exercises in Exercise01-classes-simple-textual.onb
- Do Exercises 1.3, 1.4, 1.5 and 2

Exercise 1.3 – Model the System Below

- Model this Simple System of Equations in Modelica

$$\dot{x} = 2 * x * y - 3 * x$$
$$\dot{y} = 5 * y - 7 * x * y$$
$$x(0) = 2$$
$$y(0) = 3$$

