OpenModelica and Dakota Short Course Lab Exercises

1. How to create a new Modelica package and model
2. How to create a directory-structured library
3. Simple tank – pipe – sink model
    a. Initial model with fundamental components
    b. Incorporate tank fill level calculation and valve (open, halfway closed, ramp closure at delayed time)
    c. Incorporate PID control of tank level
    d. Incorporate cascade PID control of tank level
    e. Incorporate mass flow disturbance into tank and compare single vs. cascade feedback loop responses
4. Using DAKOTA to conduct parameter studies
    a. Optimize mass flow rate into control model (single variable search)
    b. Optimize mass flow rate and PID parameters (multidimensional search)
    c. Optimize mass flow rate into control model via pattern search
5. Compare manually-determined optimal parameter values to those found with DAKOTA
6. Define single phase media using tabular properties

1. How to create a new Modelica library, with inherent directory structure
   - Right-click in the Libraries Browser pane and select New Modelica Class from the drop-down menu
     - alternatively, click the New Modelica Class icon 
   - Enter ThermoFluidsLabLib as the Name
   - Select Package from the Specialization drop-down menu
   - Using the Browse button, specify that this package will extend Modelica.Icons.Package
   - **To make sure the created library has an inherent directory structure, uncheck the "Save contents in one file" option**
     - If this step is neglected, all future work will be saved in one model file (.mo) as opposed to a directory structure split into different folders
   - Select the OK button to create the library
     - A new library called ThermoFluidsLabLib should now be visible in the Libraries Browser pane
   - Select the Save icon and browse to the desired system location
   - Via file browser, navigate to the desired system location and observe that a folder named ThermoFluidsLabLib has been created

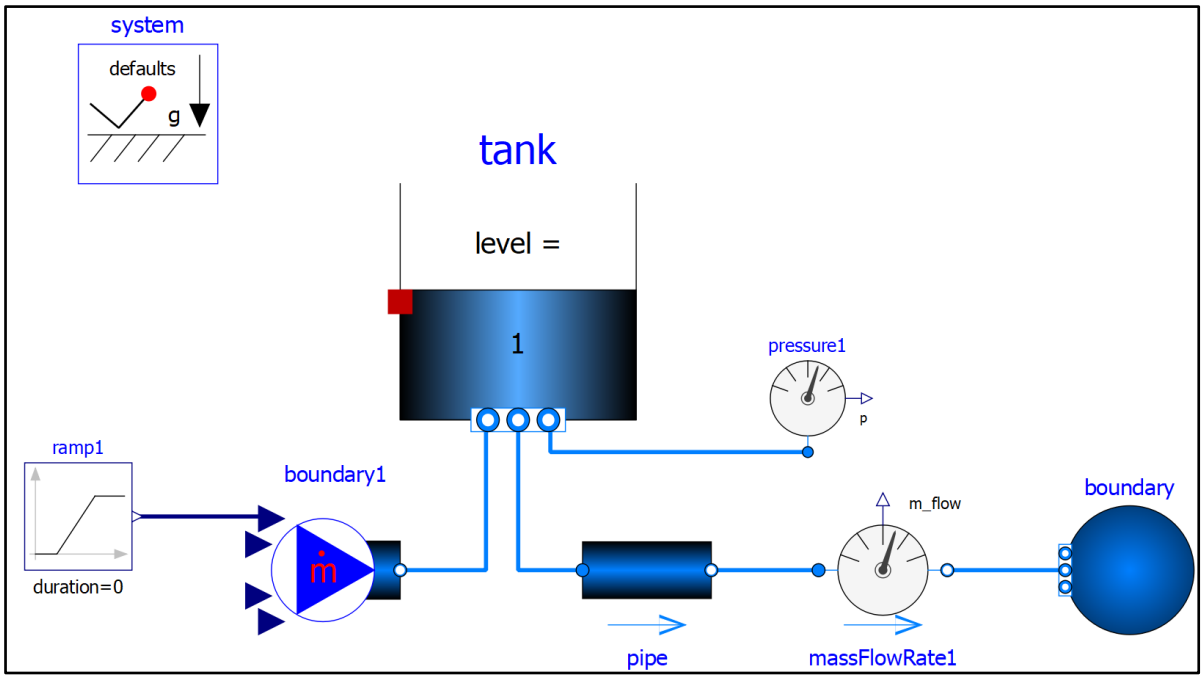
2. Creating a basic library structure
   - Following similar steps to above, the goal is to create 3 nested Modelica classes within the top-level ThermoFluidsLabLib
   - To create a nested class, right-click the library name (ThermoFluidsLabLib) and select New Modelica Class from the drop-down menu
     - If the New Modelica Class icon is selected instead, the user will be required to browse to the ThermoFluidsLabLib library for the Insert in class (optional) field
   - The table below shows the details for the three desired classes:

Name	Specialization	Extends	Save contents in one file
Components	Package	Modelica.Icons.Package	Uncheck
Examples	Package	Modelica.Icons.ExamplesPackage	Uncheck
Media	Package	Modelica.Icons.Package	Uncheck

   - Once all steps are completed, the ThermoFluidsLabLib should now have a plus sign icon next to it that when selected displays the three nested classes
   - Select the Save icon
   - Via file browser, navigate to the desired system location and observe that now the ThermoFluidsLabLib folder has become a the top level of a file directory, with three nested folders

3a. Initial model with fundamental components

- Within the Examples package, create a new Modelica class
  - Name: SimpleTankPipeFlow
  - Specialization: Model
  - Extends: Modelica.Icons.Example
  - Save contents in one file: uncheck
- Select the Diagram View of the OpenModelica Connection Editor within the Modeling tab
- Drag and drop the following components onto the diagram editor pane:
  - Modelica.Blocks.Sources.Ramp
  - Modelica.Fluid.Vessels.OpenTank
  - Modelica.Fluid.Sources.MassFlowSource_T
  - Modelica.Fluid.Pipes.StaticPipe
  - Modelica.Fluid.Sensors.Pressure
  - Modelica.Fluid.Sensors.MassFlowRate
  - Modelica.Fluid.Sources.FixedBoundary
  - Modelica.Fluid.System
- Connect the components together to create the diagram shown below
  - Connections are made by clicking the appropriate port on the source component and then clicking the appropriate port on the destination component
  - Hovering over the port on the source component should show crosshairs – clicking the appropriate port when the crosshairs are showing will begin the connection
  - As the connection is dragged toward the destination component port, the cursor will be an arrow again.
  - When the appropriate destination port is reached, the cursor should turn into crosshairs again – clicking the appropriate port will end the connection, and a line should appear which connects the source and destination ports.



- Double-click each component to launch an editor and make the following changes:
  - Modelica.Blocks.Sources.Ramp
    - height: 0
    - duration: 0
    - offset: 1
    - startTime: 0

    These changes dictate a constant mass flow of 1 kg/s into the tank.
  - Modelica.Fluid.Vessels.OpenTank
    - height: 1.5
    - crossArea: 1.5
    - nPorts: 3
    - use_portsData: false
    - Initialization > level_start: 1

    These changes specify the dimensions of the tank, how many connections it needs to make with other components, and the initial media level.
  - Modelica.Fluid.Sources.MassFlowSource_T
    - use_m_flow_in: check
    - nPorts: 1

    These changes ensure that the ramp source is used as the input mass flow profile, and specifies the number of connections to other components.
  - Modelica.Fluid.Pipes.StaticPipe
    - nParallel: 1
    - length: 1
    - diameter: 0.1

    These changes specify the dimensions of the pipe.
  - Modelica.Fluid.Sensors.Pressure
  - Modelica.Fluid.Sensors.MassFlowRate
  - Modelica.Fluid.Sources.FixedBoundary
    - nPorts: 1

    These changes specify the number of connections to other components
  - Modelica.Fluid.System
    - Assumptions > allowFlowReversal: false
    - Initialization > m_flow_start: 0.06

    These changes specify the initial mass flow rate, and that flow reversal is allowed.
- Switch to the Text View of the model
- After the extends statement, insert the following on the next line:

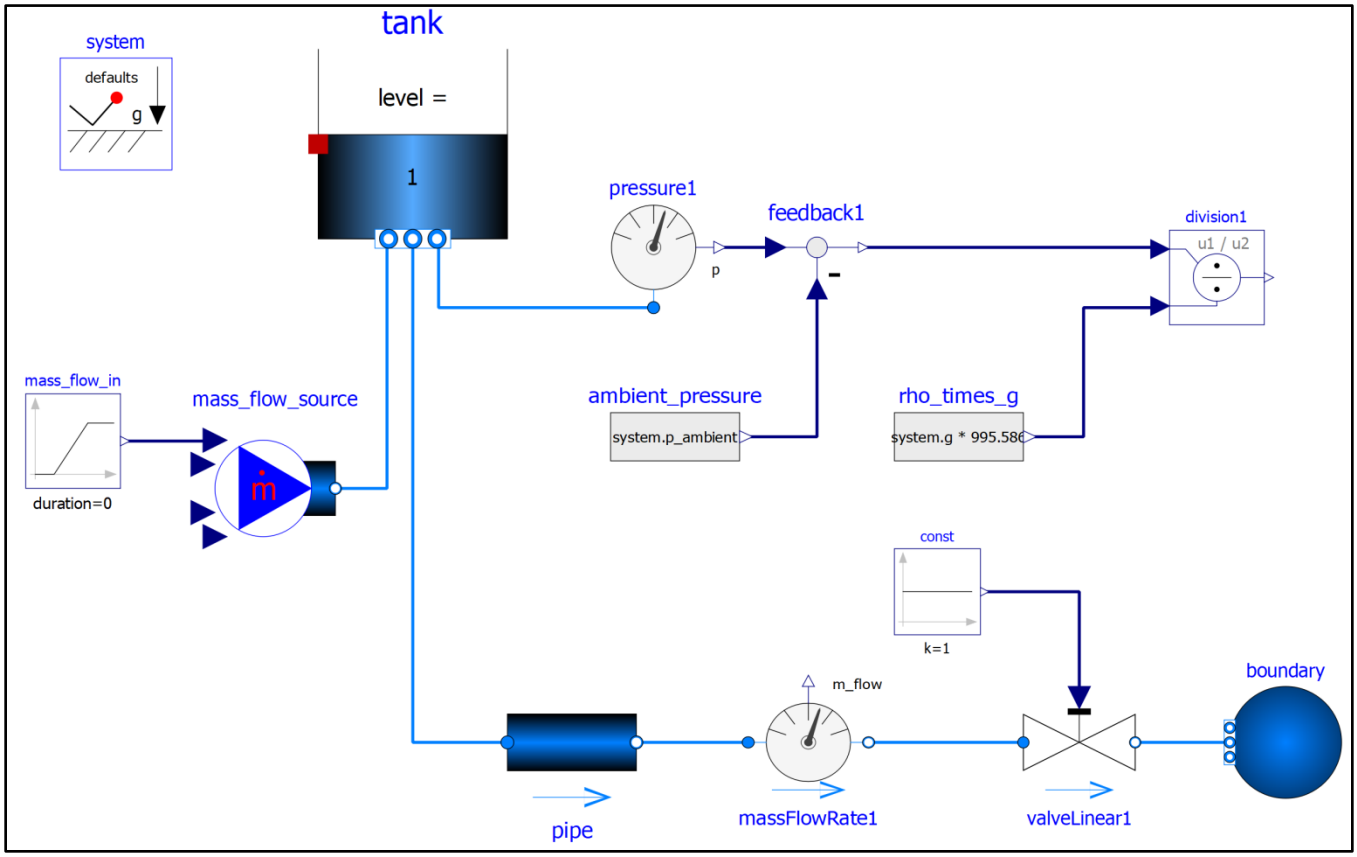
  ```
  package water = Modelica.Media.Water.ConstantPropertyLiquidWater;
  ```
- Within the parentheses of each component, insert the following:

  ```
  redeclare package Medium = water
  ```
- Save the model
- Select Check Model 
- After some time, a dialog should appear stating that the check was completed successfully
- Once the check is completed successfully, select Simulation Setup 
- In the dialog that appears, enter the following:
  - Stop Time: 1000
  - Number of Intervals: 500

- Select OK – a compilation and simulation dialog should pop-up showing the simulation progress
- Once the simulation is complete, results should automatically open within the Plotting tab
    - Selecting the + icon next to each component will expand the selectable quantities which can be visualized for it
    - Simply select the empty box next to a quantity to view its values over the simulation on the centered plot
    - Some quantities of interest:
        - tank.level
            - Notice that the initial level in the tank is indeed 1m, as set when building this model
            - The level declines quickly to 0 since the tank is simply connected to a pipe which lets the water flow out to atmospheric conditions (dictated by the boundary)
            - For future reference: A new plot can be created by selecting 

3b. Incorporate tank fill level calculation and valve

- Right-click the SimpleTankPipeFlow class in the Libraries Browser and select Duplicate
- In the dialog, enter “TankLevelCalcValve” as the new name, the path should be auto-filled with ThermoFluidsLabLib.Examples
- Navigate to the Diagram View of the generated TankLevelCalcValve class
- Drag and drop the following components onto the diagram editor pane:
    - Modelica.Blocks.Math.Feedback
    - Modelica.Blocks.Math.Division
    - Modelica.Blocks.Sources.RealExpression (x2 – need two instances of this)
    - Modelica.Fluid.Valves.ValveLinear
    - Modelica.Blocks.Sources.Constant
- Connect the components together to create the diagram shown below
    - Note: to edit the name of the components, right-click them and select Attributes



- Double-click each component to launch an editor and make the following changes:
    - Modelica.Blocks.Sources.RealExpression (ambient_pressure)
        - y: system.p_ambient

These changes provide the ambient pressure of the system, to be used in the calculation of the tank level.

- Modelica.Blocks.Sources.RealExpression (rho_times_g)
  - y: system.g * 995.586

These changes provide the density of water multiplied by the gravitational constant, to be used in the calculation of the tank level.

- Modelica.Fluid.Valves.ValveLinear
  - m_flow_nominal: 2
  - dp_nominal: 0.01

These changes specify the nominal properties of the valve.

- Modelica.Blocks.Sources.Constant
  - k: 1

These changes specify the opening of the valve – value can be between 0 and 1, with 0 indicating the valve is totally closed and 1 indicating the valve is totally open.

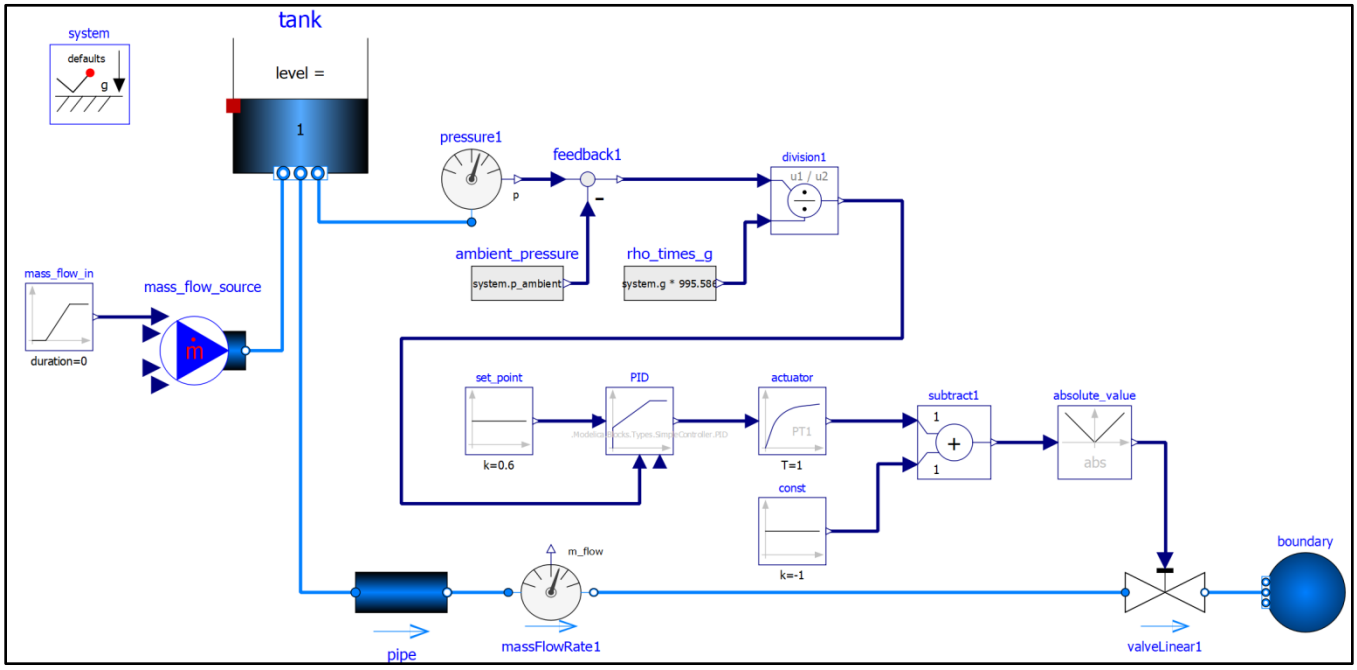
- Switch to the Text View of the model
- Within the parentheses of each component, insert the following (if necessary):

  redeclare package Medium = water
- Save the model
- Select Check Model 
- After some time, a dialog should appear stating that the check was completed successfully
- Once the check is completed successfully, select Simulation Setup 
- In the dialog that appears, enter the following:
  - Stop Time: 1000
  - Number of Intervals: 500
- Select OK – a compilation and simulation dialog should pop-up showing the simulation progress
- Once the simulation is complete, results should automatically open within the Plotting tab (reminder: a new plot can be created by selecting  in the Plotting tab)
  - Compare tank.level to division1.y – notice that the level calculation is being made correctly, exactly matching the level reported by the tank component
  - Due to the incorporation of the linear valve, the tank level now asymptotically approaches ~0.145m
- Change the value of the constant actuating the valve opening to 0.5 and run the simulation again
  - For this run, the valve is closed halfway throughout the entire simulation
  - Now the tank level asymptotically approaches ~0.29m
- Change the value of the constant once more to 0.2 and run the simulation again
  - For this run, the valve is 20% open / 80% closed throughout the simulation
  - Now the tank level approaches ~0.73m

- Conclusion: the level in the tank can be controlled by opening and closing the valve
    - Next steps: incorporate a PID controller to actuate the valve, achieving this control

3c. Incorporate PID control of tank level

- Right-click the TankLevelCalcValve class in the Libraries Browser and select Duplicate
- In the dialog, enter “TankLevelControl” as the new name, the path should be auto-filled with ThermoFluidsLabLib.Examples
- Navigate to the Diagram View of the generated TankLevelControl class
- Drag and drop the following components onto the diagram editor pane:
  - Modelica.Blocks.Sources.Constant (x2 – need two instances of this)
  - Modelica.Blocks.Types.SimpleController.PID
  - Modelica.Blocks.Continuous.FirstOrder
  - Modelica.Blocks.Math.Add
  - Modelica.Blocks.Math.Abs
- Connect the components together to create the diagram shown below



- Double-click each component to launch an editor and make the following changes:
  - Modelica.Blocks.Sources.Constant (set_point)
    - k: 0.6

    These changes specify the set level desired in the tank.
  - Modelica.Blocks.Sources.Constant (const)
    - k: -1

    For stable PID behavior, proportional gain cannot be negative. With a positive gain, the output of the PID controller and first order filter is the opposite of what is desired (opens the valve when level is below the set point, and closes it when it is above). To correct this, 1 is subtracted from the output signal which shifts the range of the values from (0, 1) to (-1, 0) and then the absolute value is taken to obtain proper valve actuation.

- Modelica.Blocks.Types.SimpleController.PID
  - yMax: 0.8
  - yMin: 0.2

  These changes specify the range of valve actuation from 0.2 to 0.8, a typical operation range for industrial valves.
- Modelica.Blocks.Continuous.FirstOrder (actuator)
  - T: 1
  - yStart: 0.5

  These changes specify that the first order filter should be applied each second, and the valve should start halfway open/closed.
- Modelica.Blocks.Math.Add
  - k1: 1
  - k2: 1

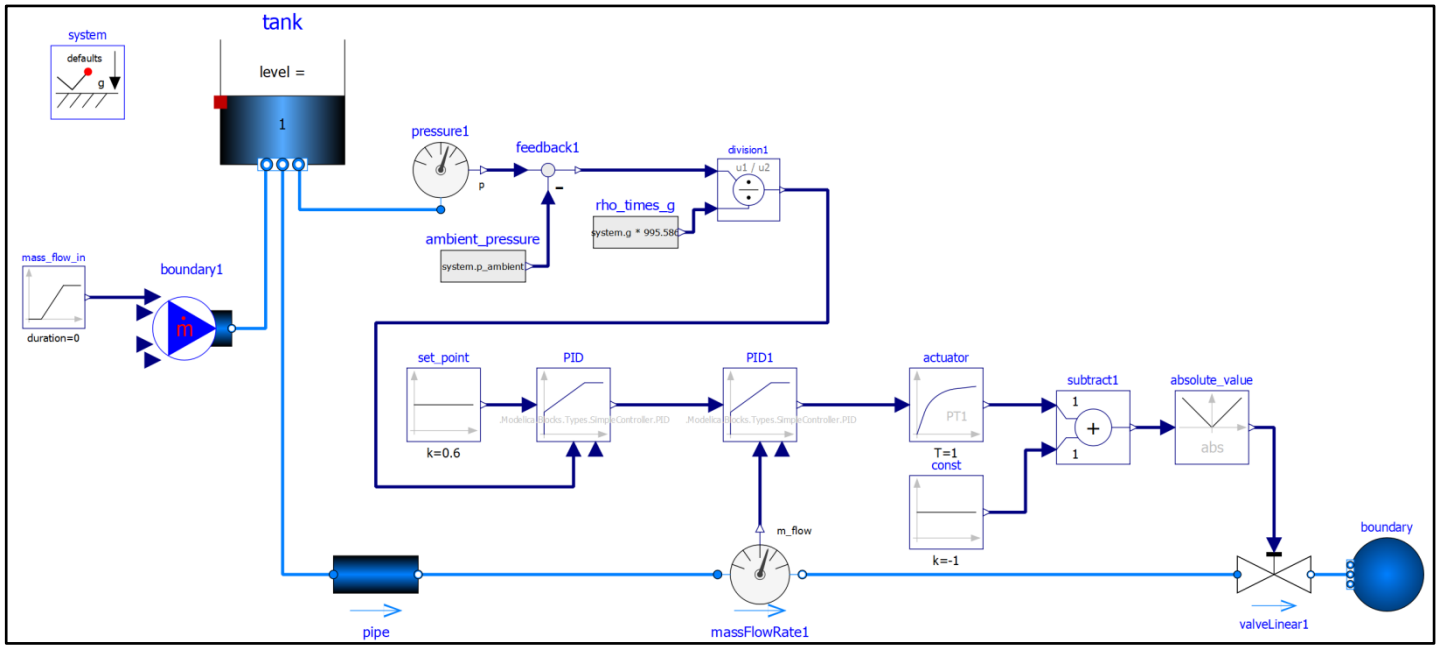
  These changes specify the gain of the first order filter and constant which are to be added (both set to 1 since no gain/reduction of either signal is needed).

- Switch to the Text View of the model
- Within the parentheses of each component, insert the following (if necessary):
  `redeclare package Medium = water`
- Save the model
- Select Check Model 
- After some time, a dialog should appear stating that the check was completed successfully
- Once the check is completed successfully, select Simulation Setup 
- In the dialog that appears, enter the following:
  - Stop Time: 1000
  - Number of Intervals: 500
- Select OK – a compilation and simulation dialog should pop-up showing the simulation progress
- Once the simulation is complete, results should automatically open within the Plotting tab
  - Look at tank.level and valveLinear1.opening
    - For the first 50s when the level in the tank is greater than the set point of 0.6m, the valve is opened as much as is allowed, to 0.8
    - Once the tank level dips below 0.6, the valve is closed as much as is allowed, to 0.2
    - The closure is maintained for the rest of the simulation since the level in the tank stays below the set point – this means that even with the valve limiting the flow as much as is allowed, there is not sufficient mass flow to maintain the desired level
      - Hypothesis: the mass flow into the system needs to be increased (it is currently set at 1 kg/s)
- Set mass_flow_in.offset to 5 kg/s and run the simulation again with csv output format

- Looking at the tank level and valve opening again reveals that this mass flow is much more suitable for maintaining a tank level of 0.6m
- Next step: what is the ideal mass flow in to sustain the set point with minimal error?
    - For now, try out a few different values, with the goal being to manually determine the optimal value
        - By eye, it seems that 2.8 kg/s is the best suited mass flow rate to maintain a level of 0.6m in the tank
    - In a later lab exercise, a quantitatively optimal value will be obtained by using DAKOTA, an open-source optimization and uncertainty quantification software

3d. Incorporate cascade PID control of tank level

- Right-click the TankLevelControl class in the Libraries Browser and select Duplicate
- In the dialog, enter "TankLevelCascadeControl" as the new name, the path should be auto-filled with ThermoFluidsLabLib.Examples
- Navigate to the Diagram View of the generated TankLevelCascadeControl class
- Drag and drop the following components onto the diagram editor pane:
  - Modelica.Blocks.Types.SimpleController.PID
- Connect the components together to create the diagram shown below



- Double-click each component to launch an editor and make the following changes:
  - Modelica.Blocks.Types.SimpleController.PID (PID1)
    - yMax: 0.8
    - yMin: 0.2

    These changes specify that the valve actuation should be between 0.2 and 0.8.
  - Modelica.Blocks.Types.SimpleController.PID (PID)
    - yMax: 15
    - yMin: 0

    These changes specify that the mass flow rate should be between 0 and 15 kg/s (determined by plotting massFlowRate1.m_flow)
- Switch to the Text View of the model
- Within the parentheses of each component, insert the following (if necessary):
  redeclare package Medium = water
- Save the model

- Select Check Model 
- After some time, a dialog should appear stating that the check was completed successfully
- Once the check is completed successfully, select Simulation Setup 
- In the dialog that appears, enter the following:
    - Stop Time: 1000
    - Number of Intervals: 500
- Select OK – a compilation and simulation dialog should pop-up showing the simulation progress
- Once the simulation is complete, results should automatically open within the Plotting tab
    - Look at the tank level and valve opening
        - With the default gain values of the PID controllers, this cascade setup does not achieve the same control as the simple control model
        - Hypothesis: proportional gain values should be adjusted
            - Further, the mass flow changes more dramatically than the level within the tank
            - This indicates that the proportional gain of the first PID should be increased (so that the slow change has more of an effect on the control) and the proportional gain of the second PID should be decreased (so that the fast change has less of an effect on the control)
- Change PID.k to 1000 and PID1.k to 0.1, then rerun the simulation
    - Better control is achieved with these proportional gain settings, but it still seems like the response of the controller to the tank level being below the set point is not strong enough
        - Hypothesis: try increasing PID.k further to improve control
- Change PID.k to 5000, then rerun the simulation
    - This gives better control of the tank level, as well as less variation in the valve actuation

3e. Incorporate mass flow disturbance into tank and compare single vs. cascade feedback loop responses

- Right-click the TankLevelControl and TankLevelCascadeControl classes in the Libraries Browser and select Duplicate
- In each dialog, enter “TankLevelControl_Disturb” and “TankLevelCascadeControl_Disturb” as the new respective names, the path should be auto-filled with ThermoFluidsLabLib.Examples
- Navigate to the Diagram View of the generated classes
- Make the following edits to the mass_flow_in component (of each model):
    - height : 3
    - duration : 0
    - offset : 2.8
    - startTime : 400
- Save the models, and run a simulation for 1000 seconds for each
- In the plotting tab, compare the results to the undisturbed models (specifically, look at tank.level for each model)
    - Without the disturbance, it was clear that the cascade control scheme was capable of more accurately maintaining the tank level
    - With the disturbance, each feedback loop responds appropriately – however, the cascade response is again more accurate and results is lower amplitude oscillations about the set point
    - If the example system had a tolerance of ±0.1m fluctuation of the tank level, it is clear that the cascade control scheme is capable of maintaining the level within that tolerance while the single feedback loop control structure is not

4. Using DAKOTA to conduct parameter studies

PRELIMINARY REQUIREMENTS: machine must have Windows OS or access to Windows Powershell

- Strongly suggest just using a Windows OS machine
- For those who want to download Windows Powershell on macOS, the files can be found at: https://github.com/PowerShell/PowerShell/releases/tag/v7.0.0

- Open the Dakota GUI (application for this can be found at:

%install_directory% \ Dakota \ gui \ DakotaUI.exe

- Create a new Dakota project, named TLC_single
  - Select File > New > Dakota Project
- Within TLC_single, import the .exe, .xml, and .csv files which correspond to the OpenModelica model intended to be optimized (for this exercise, the model of choice is TankLevelControl)
  - Files are imported in Dakota by right-clicking the generated Dakota project and selecting Import…, then File System, then navigating to the correct directory and selecting the desired files
  - Whenever OpenModelica compiles and simulates a model, it generates an executable and a .xml file from which it initializes the executable prior to execution (i.e. simulation)
  - When the simulation is complete, results are output to a file
    - This can be either .mat, .plt, or .csv
    - For this application, .csv is most convenient
  - Remark that on the last simulation run of TankLevelControl, the output format should have been changed to .csv
    - If this step was missed, go back to OpenModelica and run that model again with the appropriate parameters and output format
  - All three of these files should be located in the Working Directory of OpenModelica
    - To find this location, select Tools > Open Working Directory in OpenModelica
- Select the > next to the Dakota project name
  - The three files should now appear in a drop-down list below the Dakota project name
- For convenience/compatibility with driver, rename these files as:
  - tlc.exe
  - tlc_out.csv

Rename files by right-clicking them in the drop-down list and selecting Rename...

    - tlc_template.xml
- Navigate to the folder containing the materials for this lab and import driver_mad_sse.ps1 and driver_multi.psi
- Open both drivers (driver_mad_sse.ps1 and driver_multi.ps1; double-clicking them should open each in a test editor, e.g. Notepad) and tlc_template.xml (double-clicking it should open it in an embedded Dakota text editor pane)
    - On line 12 of the .xml file, there should be:

      guid = "{some_alphanumeric_expression}"

    - On two different lines within each .ps1 file, there should be two instances of a similar statement
- Copy the alphanumeric expression FROM the .xml file TO both appropriate locations within each .ps1 file
    - Only copy the alphanumeric portion – it is important that the surrounding text, including all spacing and formatting, remain unchanged
    - Once copied successfully, save the updated drivers (.ps1 files)
- Create a new Dakota project, named TLC_multi
    - Select File > New > Dakota Project
- Move driver_multi.ps1 to TLC_multi
    - Right-click driver_multi.ps1 and select Move…, then choose TLC_multi as the destination
- Predictive troubleshooting; ensure that the following is true for the machine being used:
    - Navigate to the Dakota preferences (the Dakota tab within the window launched by selecting Window > Preferences within the main Dakota GUI)
        - Is the path to the Dakota executable correct?
        - Is the path to an installation of Python 3.x correct?
            - If not, is Python 3.x installed at all?
                - Yes – correct the path within the Dakota preferences window and select Apply
                - No – install from https://www.python.org/downloads/, then correct the path within the Dakota preferences window and select Apply
    - Are you using a Windows machine (or have access to Windows Powershell)?
        - Yes – hooray, continue
        - No – review the preliminary requirements of this lab exercise

4a. Optimize mass flow rate into control model (single variable search)

- For the duration of the content in this subsection, all instructions pertain to the TLC_single Dakota project
- From the directory with the files for this lab exercise, import tlc_single.in
- Double-click tlc_single.in within the drop-down file list to open it in the embedded Dakota text editor
- This .in file is a Dakota study file
    - It can be created via File > New > Dakota Study, but the UI prompts to automatically generate this file necessitate the creation and inclusion of many more files in order to properly configure everything
        - in the end, this process is tedious and does not contribute anything to the study ultimately conducted
    - Dakota studies consist of 6 major components:
        - Environment
            - This specifies the kind of output file that is desired, and the format of "tabular.data" will be apparent once a study is conducted
        - Method
            - This specifies the kind of study
                - There are many different options spanning a huge range of different optimization schemes and approaches (refer to the Dakota User Manual for more information)
                - For this lab exercise, a centered parameter study will be conducted
                - This kind of study runs a series of simulations, each given a different parameter value within the provided range centered about the initial point
        - Model
            - This specifies the model, variables, and response pointers
        - Variables
            - This specifies the number of variables which will be varied, as well as defines the range over which they can be varied
            - The descriptor is incredibly important – it must appear exactly with specific formatting in the tlc_template.xml file for this study to be run (Steps for this will be provided in the next sections)
        - Responses
            - This specifies the responses of the study, or the results that will be used to determine the optimal parameter values
                - For this study, there will be two responses – mean absolute difference (mad) and sum of squared errors (sse)

        - Details of the calculation of these will be provided in the next sections
    - Interface
        - This specifies how each simulation should be run (Powershell), which files to use as input/output templates, the name of the executable which should be run, and where to store results for each simulation run (so that all results over the entire study can be viewed in one place once the study is completed)
- Nothing in the .in file needs to be changed – it has already been configured for the purposes of this lab, to cut down on the initial learning curve of Dakota syntax and semantics
- Calculation of results:
    - For this study, the control achieved by a PID controller is to be optimized
    - At each point of time in the simulation, the PID component in OpenModelica reports an error quantity, PID.controlError, as:

      PID.controlError = set point – measurement
    - So, for a simulation lasting 1000 seconds, OpenModelica produces a vector called PID.controlError which contains 1000 difference values
    - This vector contains too much information to easily make a decision as to which parameter values are optimal; summary statistics for each simulation are needed
        - Mean absolute difference:

          $mad = \sum abs(PID.controlError[i])$
        - Sum of squared errors:

          $sse = \sum (PID.controlError[i])^2$
    - These calculations are made at the end of the driver (.ps1 file), in PowerShell syntax
- Editing the template file
    - Open tlc_template.xml
    - Find (ctrl + F) “variableFilter”
        - This complete line should be:

          `variableFilter      = ".*" />`
        - Replace .* with PID.controlError
        - The complete line should now be:

          `variableFilter      = "PID.controlError" />`
    - Find (ctrl + F) “offset”
        - This should locate the ScalarVariable named mass_flow_in.offset
        - This is the variable that will be changed by during the Dakota study
        - On the last line of information pertaining to this variable, the complete line should be:

          ```
          <Real start="5.0" fixed="true" useNominal="false" />
          ```
        - Change this to:

```
    <Real start="{mdot = 2.5}" fixed="true" useNominal="false" />
```

- Save the updated .xml file

- Run the Dakota study
  - Right-click tlc_single.in and select Run As > Dakota
  - Running the study will generate different simulations, iterating by 0.1 kg/s through different values of the mass flow rate ($0 \leq \dot{m} \leq 5$ kg/s) into the talk level control model and recording the error statistics
- When the study has successfully completed, a new folder called run_results containing another new folder dated and timestamped according to the study run
- Within the dated and timestamped folder, there is a tabular.data file
- Open the tabular.data file to see the results of the study
  - Each value of $\dot{m}$ which was tried is accompanied by the corresponding mad and sse for that simulation run
  - The results should indicate that $\dot{m} = 2.6$ kg/s results in the lowest sse, while $\dot{m} = 2.7$ kg/s results in the lowest mad

4b. Optimize mass flow rate and PID parameters (multidimensional search)

- For the duration of the content in this subsection, all instructions pertain to the TLC_multi Dakota project
- The driver for this lab exercise, driver_multi.ps1, should have already been imported, edited, and then moved to TLC_multi as part of the steps provided under the "using DAKOTA to conduct parameter studies"
    - If this is not the case, go back to the referenced section and make sure the appropriate edits to the guid are made within driver_multi.ps1
- Within TLC_multi, import the .exe and .csv files which correspond to the OpenModelica model intended to be optimized (for this exercise, the model of choice is TankLevelControl)
- For convenience/compatibility with driver, rename these files as:
    - tlc.exe
    - tlc_out.csv
- From the directory containing files for these lab exercises, import tlc_multi.in
    - This will be the DAKOTA study file for this lab exercise – it should be quite similar to tlc_single.in
    - As in the previous exercise, no further edits need to be made to this file – it has already been properly configured for this exercise
- The final component needed to run the DAKOTA study specified by tlc_multi.in is a new template input file where the multiple variables to be changed are properly configured
    - Duplicate (via Copy/Paste) tlc_template.xml used in the last lab within the TLC_single project, and name the new file tlc_template_multi.xml
    - Move tlc_template_multi.xml to the TLC_multi project and open it
    - Find (ctrl + F) "PID.gainPID.k"
        - This should locate the ScalarVariable named PID.gainPID.k
        - This is one of variables that will be changed during the Dakota study
        - On the last line of information pertaining to this variable, the complete line should be:

```
<Real start="1.0" fixed="true" useNominal="false" unit="1"/>
```

        - Change this to:

```
<Real start="{k = 1.0}" fixed="true" useNominal="false" unit="1"/>
```

    - Find (ctrl + F) "PID.Ti"
        - This should locate the ScalarVariable named PID.Ti
        - This is one of variables that will be changed during the Dakota study
        - On the last line of information pertaining to this variable, the complete line should be:

```
<Real start="0.5" fixed="true" useNominal="false" min="1e-060" unit="s" />
```

- Change this to:

```
<Real start="{Ti = 0.5}" fixed="true" useNominal="false" min="1e-060" unit="s" />
```

- Find (ctrl + F) "PID.Td"
  - This should locate the ScalarVariable named PID.Td
  - This is one of variables that will be changed during the Dakota study
  - On the last line of information pertaining to this variable, the complete line should be:

```
<Real start="0.1" fixed="true" useNominal="false" min="0.0" unit="s" />
```

- Change this to:

```
<Real start="{Td = 0.1}" fixed="true" useNominal="false" min="0.0" unit="s" />
```

- Save the updated tlc_template_multi.xml

- Run the multidimensional DAKOTA study
  - Right-click tlc_multi.in and select Run As > Dakota from the drop-down menu
  - The DAKOTA study that is set up will vary four parameters as follows:
    - $2.6 \leq \dot{m} \leq 2.7 \ \frac{kg}{s}$, the mass flow rate into the tank level control model
      - This range was determined as optimal by the results of the single variable parameter study since 2.6 corresponded to the smallest sse value, and 2.7 corresponded to the smallest mad value
    - $1 \leq k \leq 5$, the proportional gain of the PID controller
    - $1 \leq Ti \leq 10$, the time constant of the integral term of the PID controller
    - $0 \leq Td \leq 3$, the time constant of the derivative term of the PID controller
  - The full study will take some time to run (10+ minutes, depending on the specifications of the machine being used)
- When the study has successfully completed, a new folder called run_results containing another new folder dated and timestamped according to the study run
- Within the dated and timestamped folder, there is a dakota_tabular.dat file
- Open the dakota_tabular.dat file to see the results of the study
  - Each value of $\dot{m}, k, Ti,$ and Td which was tried is accompanied by the corresponding mad and sse for that simulation run

4c. Optimize mass flow rate into control model via pattern search

- Create a new Dakota project named TLC_pattern
- Within this new project, import the .xml, .csv, and .exe from the appropriate OpenModelica working directory for the TankLevelControl model
- Rename them to tlc.exe, tlc_out.csv, and tlc_template.xml
- From the directory containing files for these lab exercises, import tlc_pattern.in and driver_pattern.ps1
- Using the .xml file, edit the guid in the driver file and save
- Within the tlc_template.xml, make sure the outputFormat is csv, the variableFilter is PID.controlError, and mass_flow_in.offset has the proper formatting
    - Original: `<Real start="2.5" fixed="true" useNominal="false" />`
    - Updated: `<Real start="{mdot = 2.5}" fixed="true" useNominal="false" />`
- Run the Dakota study defined by tlc_pattern.in
    - The information printed to the console will state the best parameter value identified, the corresponding objective function values (mad and sse in this case), and which iteration produced these values

5. Compare manually-determined parameter values to those found with DAKOTA
    - Return to the ThermoFluidsLabLib library within OpenModelica
    - Open the TankLevelControl class, set the mass_flow_in.offset to be 2.8 kg/s and run a simulation for 1000s
    - Duplicate (right-click, then select Duplicate) the TankLevelControl class and name the new class TankLevelControl_Dakota, saving it within the Examples class
    - In the new model TankLevelControl, set the mass_flow_in.offset to 2.6 kg/s and 2.7 kg/s (the optimal values found with DAKOTA), and compare with 2.8 kg/s in the original model
        - In the plotting tab, since there are two distinct models it should be straightforward to plot tank.level and valveLinear1.opening for the two different models and compare the results

6. Define single phase media using tabular properties
    - For the purposes of this lab exercise, consider mineral oil (Essotherm 650)
    - This comes defined as part of the Modelica Standard Library, under Modelica.Media.Incompressible.Essotherm650
        - Navigate to this model and open its text view
    - Remark that in order to define a custom incompressible single phase media, all that need be provided is a temperature range as well as tables of properties covering this range
        - Modelica will interpolate properties throughout simulation from these tables
        - From Modelica Standard Library documentation, the minimal data that needs to be provided for a useful medium description is tables of density and heat capacity as functions of temperature
    - For a sanity check on the values provided for Essotherm 650, refer to: https://www.engineeringtoolbox.com/temperature-density-petroleum-lubricating-oil-lubricant-volume-correction-ASTM-D1250-d_1943.html
    - This approach could be extended to a custom desired media, given that tabulated forms of the necessary properties are accessible and properly formatted.