

Workshop

**Object-Oriented Programming with LabVIEW**

[Creative Commons Lizenzvertrag](http://creativecommons.org/licenses/by-nc-sa/3.0/de/)  
Workshop: Objekt-orientiertes Programmieren mit LabVIEW von Dr. Holger Brand steht unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 3.0 Deutschland Lizenz](http://creativecommons.org/licenses/by-nc-sa/3.0/de/). Über diese Lizenz hinausgehende Erlaubnisse können Sie unter [tt-info@gsi.de](http://creativecommons.org/choose/tt-info@gsi.de) erhalten.

Dr. Holger Brand

GSI Helmholtzzentrum für Schwerionenforschung GmbH

Table of contents

[UPN-Calculator 3](#_Toc345493738)

[Exercise 1: Create project from template 4](#_Toc345493739)

[Project structure 5](#_Toc345493740)

[Start the Application 6](#_Toc345493741)

[Exercise 2: Evaluation of the Application 8](#_Toc345493742)

[Exercise 3: Create the class *Stack* 9](#_Toc345493743)

[Exercise 4: Extend *UPNCalculator Actor* 15](#_Toc345493744)

[Exercise 5: Extend *Calculator* Actor 19](#_Toc345493745)

[Exercise 6: Extend *UPNCalculatorGUI Actor* 22](#_Toc345493746)

[Exercise 7: Design Pattern Composition 27](#_Toc345493747)

[Exercise 7: Advanced *UPNCalculator* (Inheritance) 29](#_Toc345493748)

[Notice 31](#_Toc345493749)

[Notice 32](#_Toc345493750)

[Notice 33](#_Toc345493751)

# UPN-Calculator

An UPN-Calculator (Umgekehte Polnische Notation, Reveresed Polish Notation) uses a *Stack*, It pushes the operands of an operation on top of a stack first. When executing an operation, the necessary numbers of operands are popped from the top of the stack, the operation becomes executed and the result is pushed back to the top of the stack. The flow chart is shown below.

Type number

Execute operation

Press enter

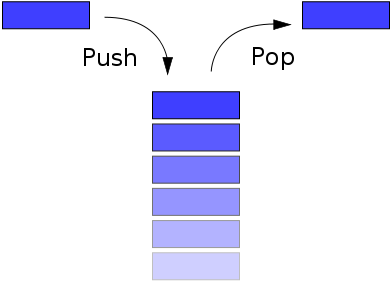
Operation executable?

no

yes

Flowchart of UPN

A stack implements a Last-In-First-Out memory structure (LiFo). The element add last will be removed first.

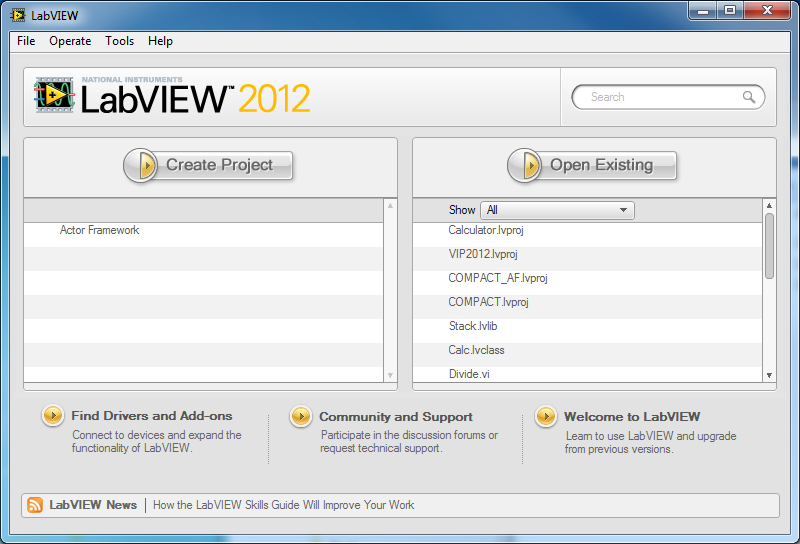
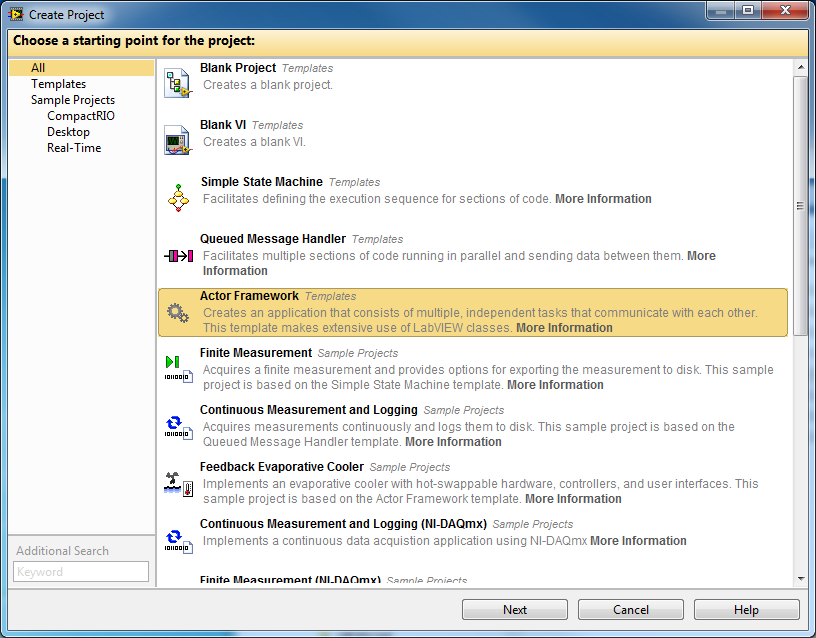
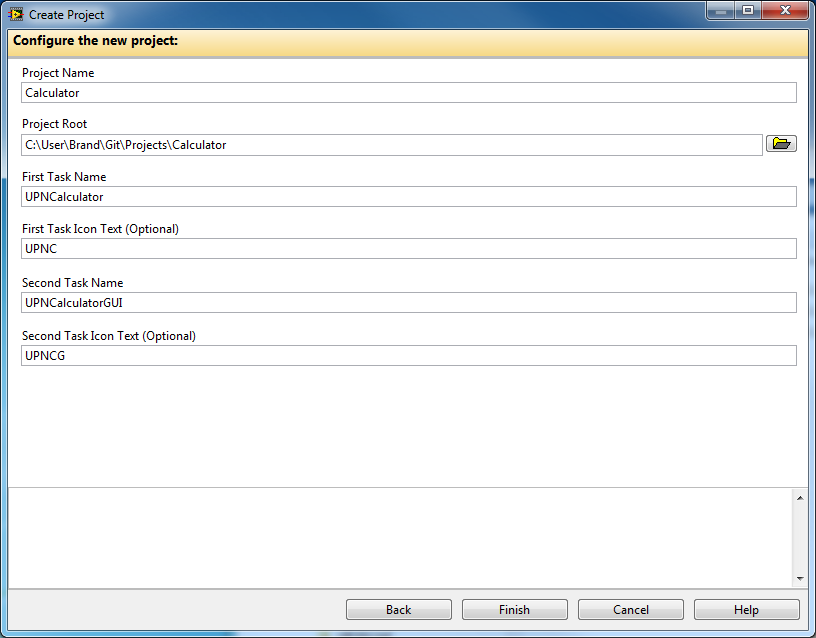
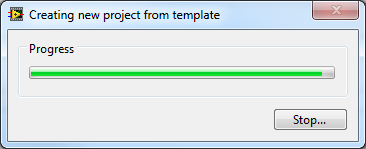
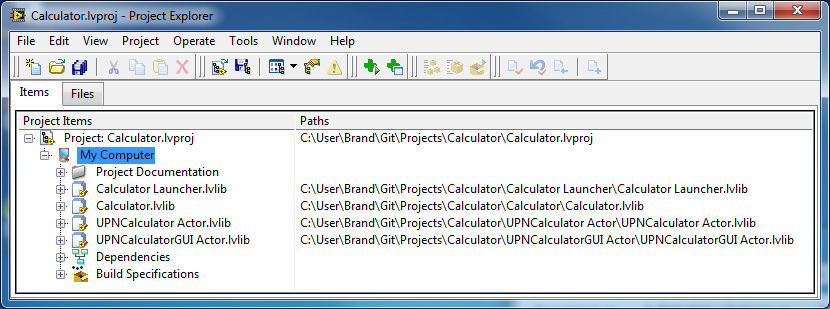
[](http://upload.wikimedia.org/wikipedia/commons/2/29/Data_stack.svg)

Schema of Stack (http://de.wikipedia.org/wiki/Stapelspeicher)

# Exercise 1: Create project from template

In this exercise you will create a new Actor Framework project from template by using the project wizard. Click *Create Project* in the LabVIEW *Getting Started Window* and select the *Actor Framework* Template. Follow the instructions to configure the template. The wizard will create some libraries, e.g. Splash Screen that will start the actual application which means activating an actor. Finally you get an executable LabVIEW project which serves as a starting point for your own developments.

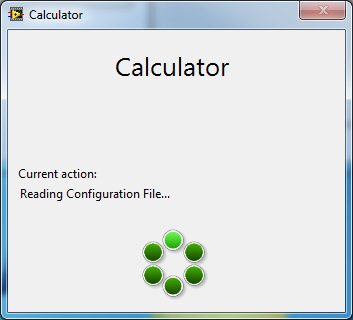
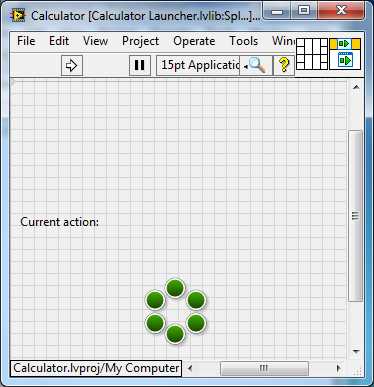
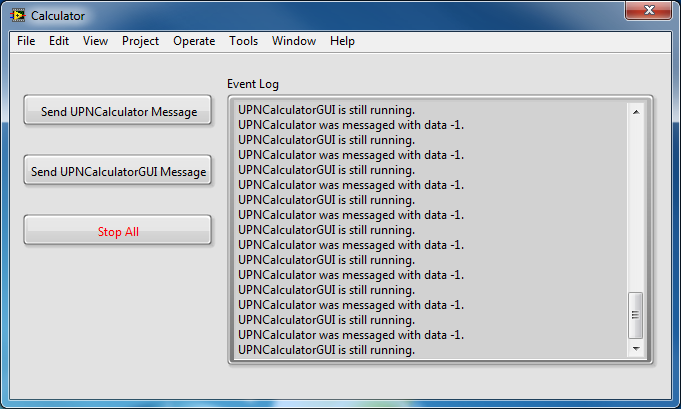
* 1. Start LabVIEW.

1. Create Project.  
    
2. Select the *Actor Framework* template and click n*ext*.  
   
3. Enter Project Name: **Calculator**. Select a suitable folder for your project fill the other fields as shown. Then click *Finish*.  
   
4. Wait until the project has been generated.  
   
5. The *Calculator* Project should now look like this:  
   

## Project structure

1. *Project Documentation* contains two important documents:
   1. *Actor Framework Whitepaper.html* contains concepts and background information
   2. *Actor Framework.html* contains information about this template.
2. *Calculator Launcher.lvlib* contains the *Splash Screen.vi* that starts the actual application which means launching an actor.
   1. *Localization.vi* provides translations of text in different languages. It is includes in all prepared actor classes. It will not be used explicitly in these exercises.
3. The other LabVIEW libraries contain actor classes and corresponding message classes that were generated by the wizard.
   1. *Calculator.lvlib* contains the *Calculator.lvclass*. An instance of this class becomes launched from the *Splash Screen.vi*. This is the actual application, it starts two *Actors*:
      1. *UPNCalculator Actor.lvlib:UPNCalculator.lvclass*  
         This class is used to implement the UPN-Calculator. It sends massages to itself periodically and provides no GUI.
      2. *UPNCalculatorGUI Actor.lvlib:UPNCalculatorGUI.lvclass*  
         This class is used to implement the GUI for the UPN-Calculator. It already provides an event loop.

## Start the Application

1. Expand *Calculator Launcher.lvlib* by clicking on the + sign.
2. Open *Splash Screen.vi* and run the VI. The class *Calculator.lvclass* becomes loaded and instance of will become launched.  
    
3. The window of the Calculator-Actors look like this:  
   
   1. Two more actors become started in the background, *UPNCalculator* und *UPNCalculatorGUI*. Both are sinding periodically messages as heartbeat to the *Calculator* actor.
   2. Click *Send UPNCalculator Message* or *UPNCalculatorGUI Message*, to dispatch messages to the associated actors. The addressed actor will asynchronously reply a message back to the Calculator, which writes it to the *Event Log*.
   3. Click *Stop All* to stop the application.

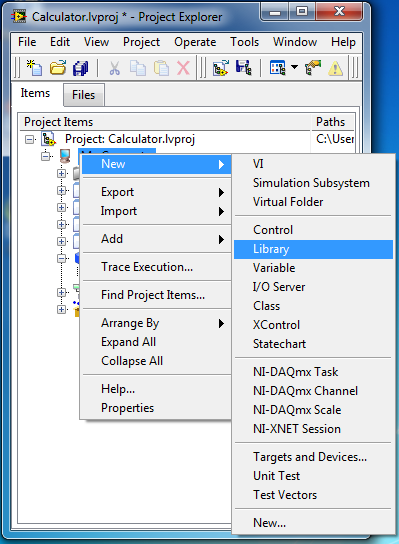
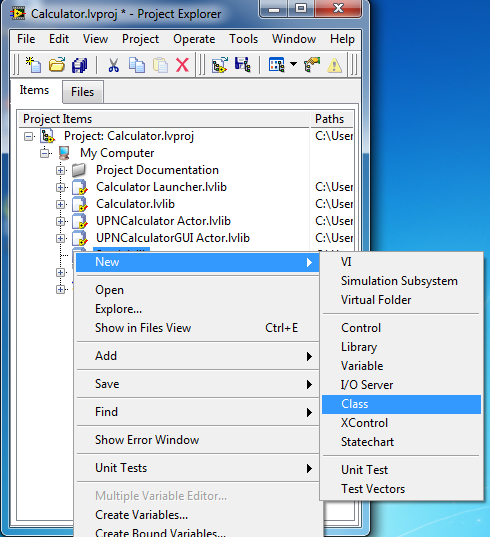
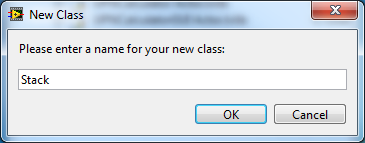
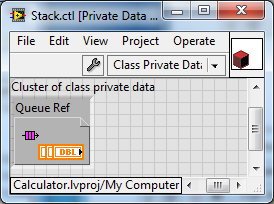
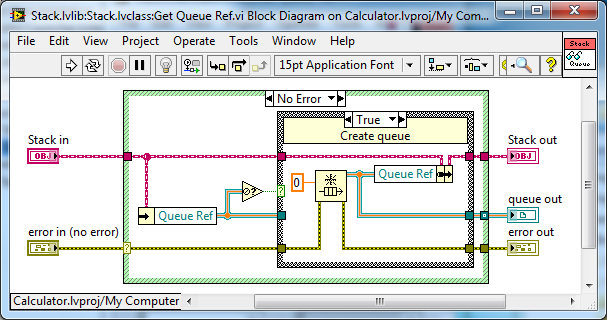
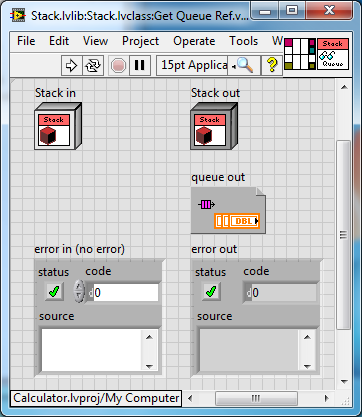
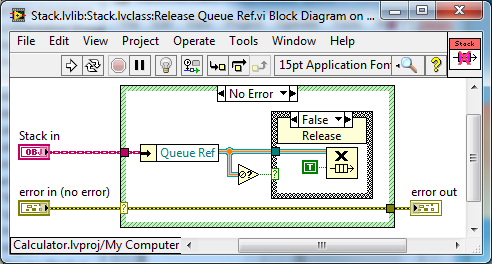
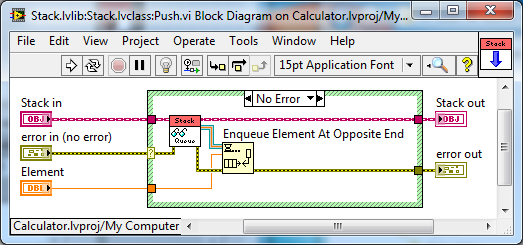
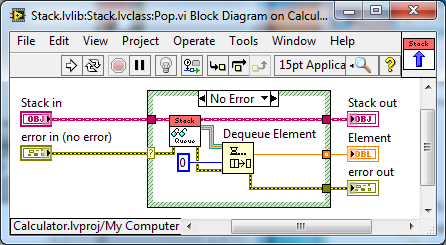
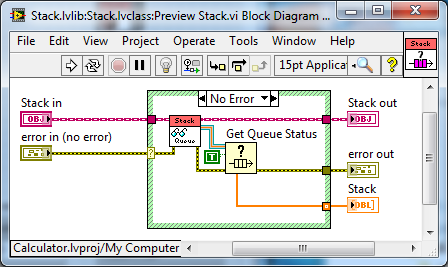
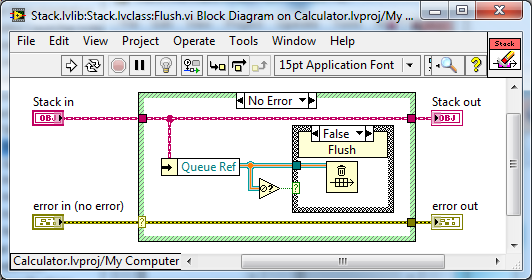
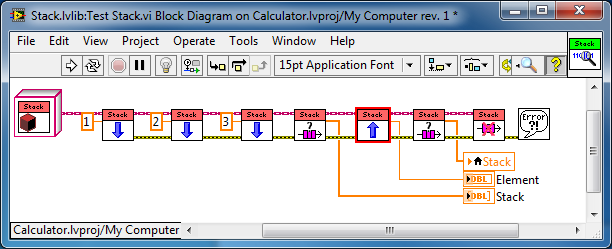
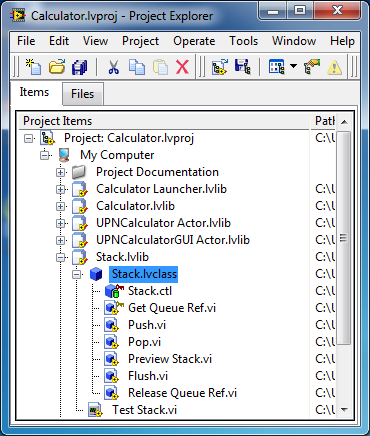
# Exercise 2: Evaluation of the Application

In this exercise we take a closer look at the application.

* I will show how the *Actor Framework Template* works and at which point the user need to modify the software to adjust and/or extend it to his needs.

# Exercise 3: Create the class *Stack*

In this exercise a queue is used to implement a stack.

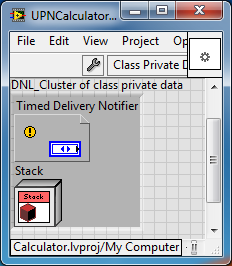
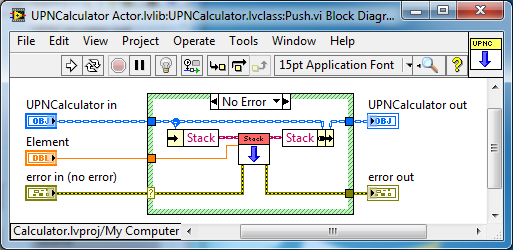
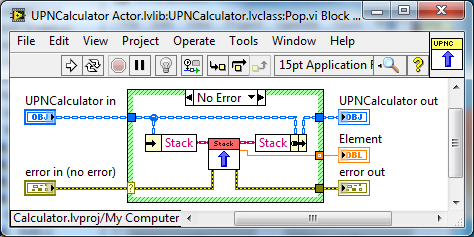
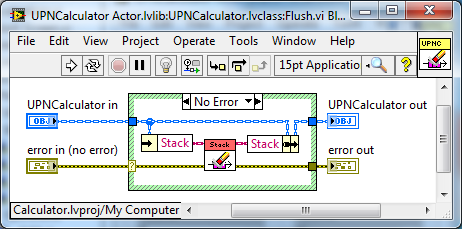
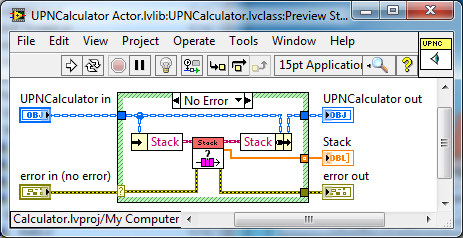
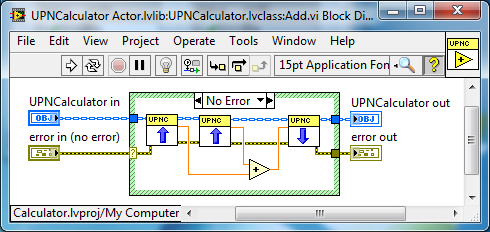
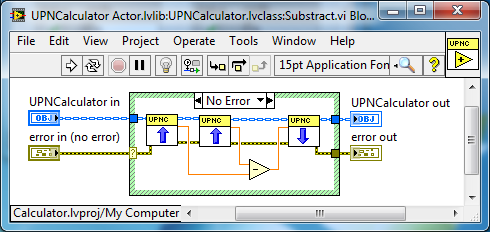
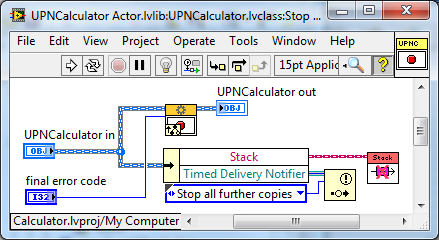
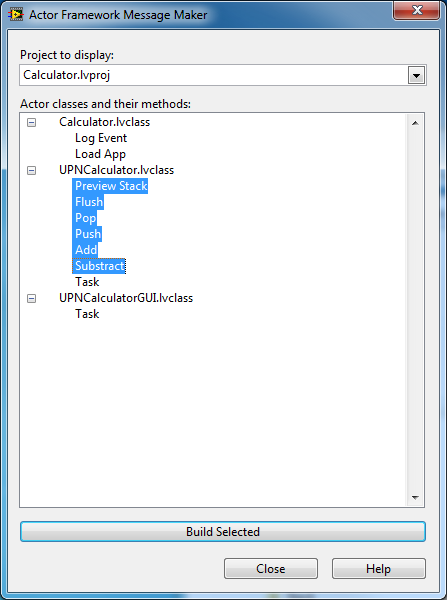
1. Create a new library *Stack*:  
   
2. Context menu of library *Save->Save All (this Library)*  
   Select a new folder in your project tree on disk.
3. Select properties in the context menu of *Stack.lvlib*.
   1. In category *General Settings* you should edit the icon.   
      Write *Stack* in line 1 of icon text.
   2. Provide a short description in category *Documentation*.
4. Create new class *Stack*: Contect menu of *Stack.lvlib->New->Class*  
   
5. Context menu of new class Save->Save All (this Class)  
   Select a new folder in your project tree on disk.
6. Select properties in the context menu of *Stack.lvclass*.
   1. In category *General Settings* you should edit the icon.   
      Write *Stack* in line 1 of icon text.
   2. Provide a short description in category *Documentation*.
7. Open Control *Stack.ctl* and add a *Queue*-Referenz for data type *double*. Save and close the control.  
   
8. Use the context menu of class to create a new *static dispatch*-VI and save it to *Get Queue Ref.vi*.  
     
   This VI should return the reference to the *queue* stored in the attributes. If the queue reference is not valid, a *queue* will be created and the reference stored in the attributes for future use. **Set *Access Scope* to *Protected*!**
9. Create a new *static dispatch*-VI and save it to *Release Queue.vi*.  
   
10. Create a new *static dispatch*-VI and save it to *Push.vi*.  
    
11. Create a new *static dispatch*-VI and save it to *Pop.vi*.  
    
12. Create a new *static dispatch*-VI and save it to *Preview Stack.vi*.  
    
13. Create a new static dispatch-VI and save it to *Flush.vi*. *Get Queue Ref.vi* is not used herein since creating a queue unnecessarily should be avoided.  
    
14. Create a new VI and save it to *Test Stack.vi*. Test the functionality of *Stack*.  
    
15. *Stack*-library should look similar to this:  
    

# Exercise 4: Extend *UPNCalculator Actor*

In this exercies you will extend the *UPNCalculator* class with an instance of *Stack* and add some calculations. The *Actor Framework Message Maker* is used to generatet he corresponding messages.

Usage of **Delegation Pattern:**

The class *Stack* is reusable. The *UPNCalculator* class delegates the stack manipulation to an object of this class. It is invisible for the user of *UPNCalculator* class.

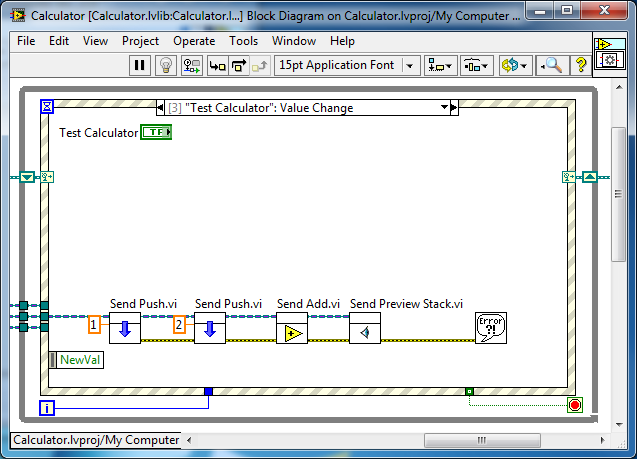
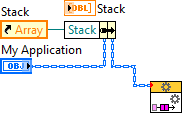
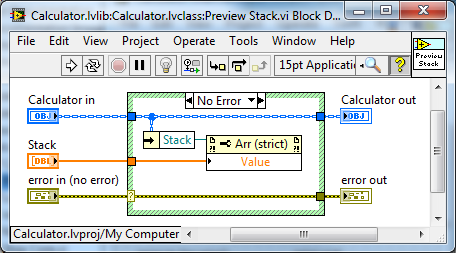
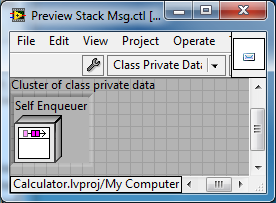
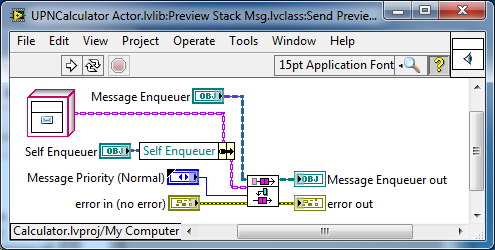
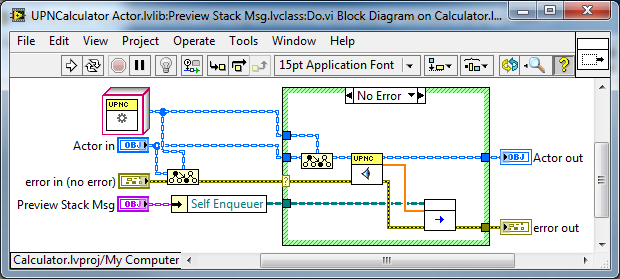
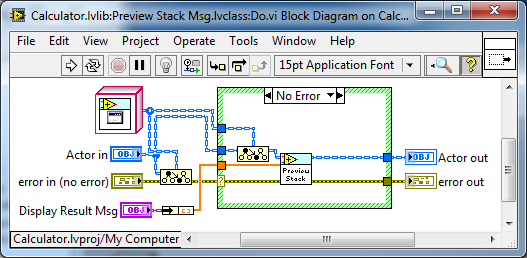
1. Open *UPNCalculator.ctl*. Add an object of class *Stack* to the attributes and close *Control*.  
   
2. Add *Wrapper*-VIs (*static dispatch*) for the *Stack* methods, *Push*, *Pop*, *Flush* and *Preview Stack*. Create a V*irtual Folder* in *UPNCalculator* class for this purpose.   
     
     
     
   
3. Create a new folder *Operations* for *static dispatch*-VIs for the operations *Add* and *Subtract*.  
     
   
4. Of course you can add more operation to this class or extend this class in a derived class.
5. Modify *Stop Core.vi* in order to release the *Queue* used by *Stack[[1]](#footnote-1)*.  
   
6. Use *Tools->Actor Framework Message Maker…* to generate the corresponding messages.  
     
   Select the desired VIs and click *Build Selected*. When the wizard finished you can close the *Message Maker*.
7. Save and close all VIs generated by the wizard.
8. Move the new *Message* classes into the virtual folder: *Messages for UPNCalculator*. Objects of this message classes become sent from the GUI actor to the calculator actor.

The *UPNCalculator* class is now prepared to reseive massages in ordert o push numbers onto the stack and to execute calculations using popping numbers from the stack.

**Exception**: *Preview Stack Msg.lvclass* must be modified to send the result of a calculation back to the GUI to be displayed. But, before the corresponding message of the *UPNCalculatorGUI* class needs to be created.

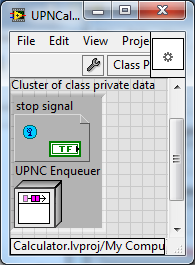
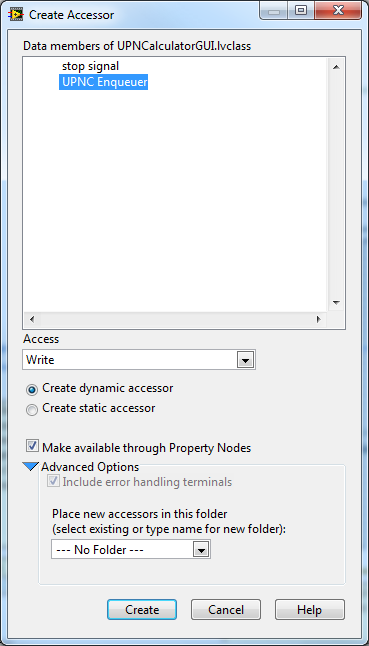
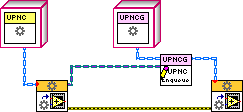
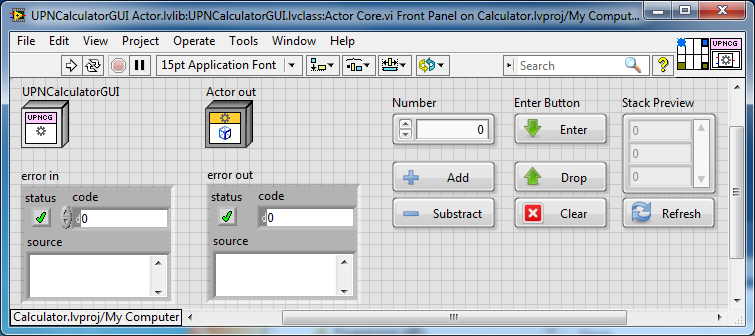
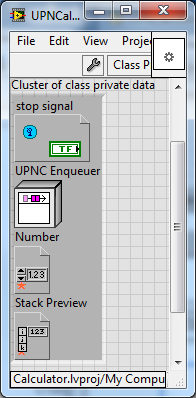
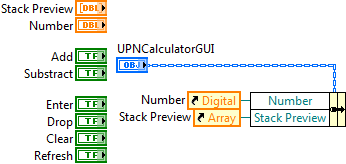
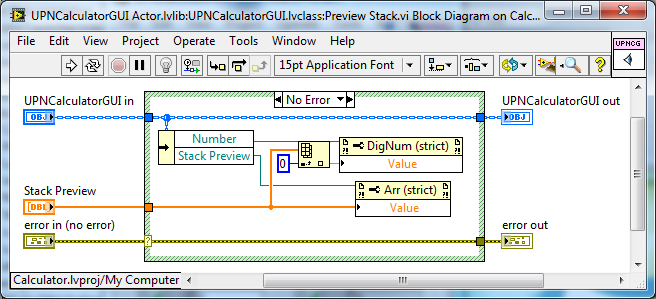
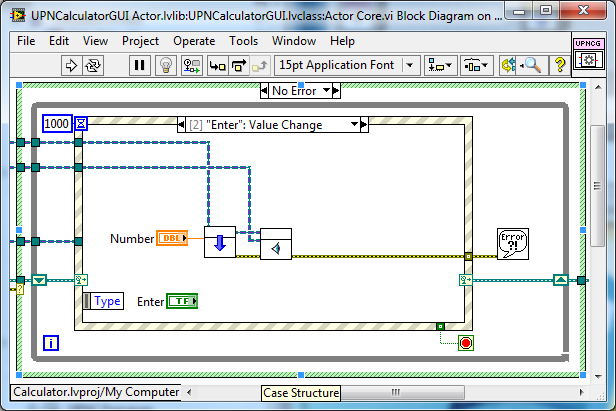
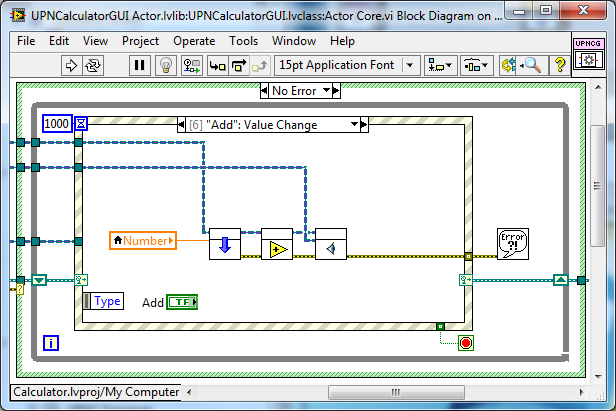
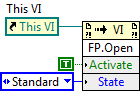
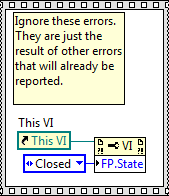
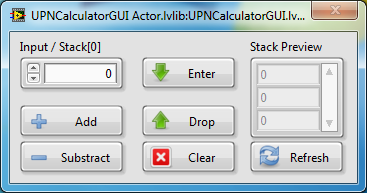
# Exercise 5: Extend *Calculator* Actor

In order o use *UPNCalculator* for calculations in a first step, add an additional event-case to *Calculator* class.

1. Open Calculator.lvclass:Actor Core.vi.  
   The actors *UPNCalculator* and *UPNCalculatorGUI* become started before *Call Parent method*, which received messages, is called. In parallel an event loop is started to handle programmatic and front panel events.
2. Place a new button on the front panel and label it *Test Calculator*.
3. Create a new event-case for this button.
4. Insert following VIs into this c*ase*:
   1. *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Push.vi*
   2. *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Push.vi*
   3. *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Add.vi*
   4. *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Preview Stack.vi*
   5. *Simple Error Handler.vi*
5. The resulting *event-case* should look similar:  
   
6. Save and close VI.
7. Start application: *Calculator Launcher.lvlib: Splash Screen.vi*
   1. Force a calculation by clicking the new button.
   2. In order to view the result of the calculation you need to open   
      *UPNCalculator Actor.lvlib:UPNCalculator.lvclass:Preview Stack.vi*.
8. Add a *Double-Array* indicator on the front panel in order to display the stack content.
9. Create a reference-*control* of the *Stack* indicator and add it to the class attribute.   
   Store the reference in the object wire before anything else.   
   
10. Create a *static dispatch*-VI to write to the indicator via *Property-Node*.   
    Save it as *Preview Stack.vi*.  
    
11. Generate the corresponding message using the *Actor Framework Message Maker*.
    * 1. Add a *Self Enqueuer* object to the attribute of *Preview Stack Msg.lvclass* in *UPNCalculator Actor.lvlib*. (Cretae a control of the output of the *Read Self Enqueuer.vi* in the AF function palette.)  
           
         The Enqueuer-Object is necessary since *UPNCalculator.lvclass:Preview Stack.vi* returns the Stack only. It does not know, and should not know what to do with it[[2]](#footnote-2). The Message-Object is responsible. The Message-Dispatcher need to know what to do with the Stack. In the simple case the Stack is send back to the Message-Dispatcher. He need to provide a corresponding method. But, the Stack cold be send to another actor, too.
12. Modify *Preview Stack Msg.lvclass:Send Preview Stack.vi* to set the *Self Enqueuer*.   
      
    **Set *Self Enqueuer* on *Connector-Pane* to *requiered*!**
13. Modify *Preview Stack Msg.lvclass:Do.vi*, to send the *Stack* content with *Calculator.lvlib:Preview Stack Msg.lvclass:Send Preview Stack.vi* back to *Self Enqueuer*.  
    
14. Modify *Calculator.lvlib:Preview Stack Msg.lvclass:Do.vi*, to display the *Stack content* on the front panel.  
    
15. Start application again: *Calculator Launcher.lvlib: Splash Screen.vi*. Observe the result.

# Exercise 6: Extend *UPNCalculatorGUI Actor*

During this exercise you will extend the *UPNCalculatorGUI* class with a reference to *UPNCalculator (Design Pattern: Aggregation)*, number input and display as well as buttons to send necessary messages to the *UPNCalculator*. You add methods and messages that allow receiving messages and to manipulate and display the stack content.

1. Extend the attribute of *UPNCalculatorGUI c*lasse with the enqueuer of the *UPNCalculator*. This object is used to send messages to *UPNCalculator Actor*.  
   
2. From the context menu of class create a new accessor *VI* for *UPNC Enqueuer*.   
   
3. Set *UPNC Enqueuer* in *Calculator.lvlib:Calculator.lvclass:Actor Core.vi*.  
   
4. Open *UPNCalculator.lvclass:Actor Core.vi*. Place necessarycalculator controls and indicators on the front panel:  
   
   1. A numerisches control for number input
   2. Buttons for stack manipulation:
      1. Enter (Push an element to Stack)
      2. Drop (Pop an element from Stack)
      3. Clear (Flush Stack)
      4. Refresh (Update Stack Preview)
   3. Buttons for calculations:
      1. Add
      2. Subtract
5. Add references to *Number* und *Stack Preview* to the attribute and set them on the block diagramm tot he object wire before anything else.  
    
6. Create a *static dispatch*-VI, to set *Stack* content via *property-node*.   
   Save as *Preview Stack.vi*.  
   
7. Generate the corresponding message using the *Actor Framework Message Maker*.
8. Replace in *UPNCalculator Actor.lvlib:Preview Stack Msg.lvclass:Do.vi* the *Calculator.lvlib:Preview Stack Msg.lvclass:Send Preview Stack.vi* with *UPNCalculatorGUI Actor.lvlib:Preview Stack Msg.lvclass:Send Preview Stack.vi*.
9. Open *UPNCalculator.lvclass:Actor Core.vi*. Add event-cases fort he new GUI buttons and dispatch the corresponding messages to the *UPNCalculator*.  
   1. **Enter:**  
      
   2. **Add:** etc.
10. Do not forget to open and close the front panel programmatically at the beginning and in the end of VI execution:  
    Am Beginn:  Am Ende: 
11. Configure the window appearance to your needs.
12. Start application again: *Calculator Launcher.lvlib: Splash Screen.vi*
13. Use the *UPNCalculatorGUI* for calculations.

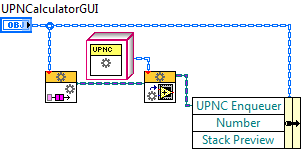
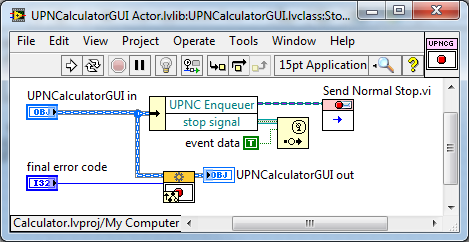
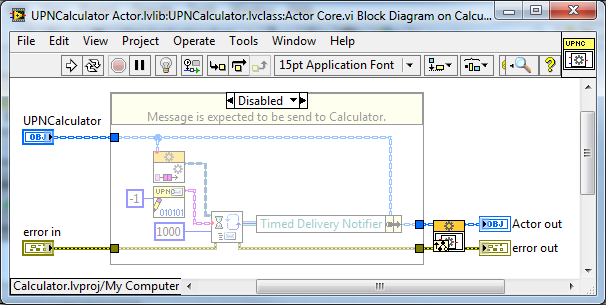
# Exercise 7: Design Pattern Composition

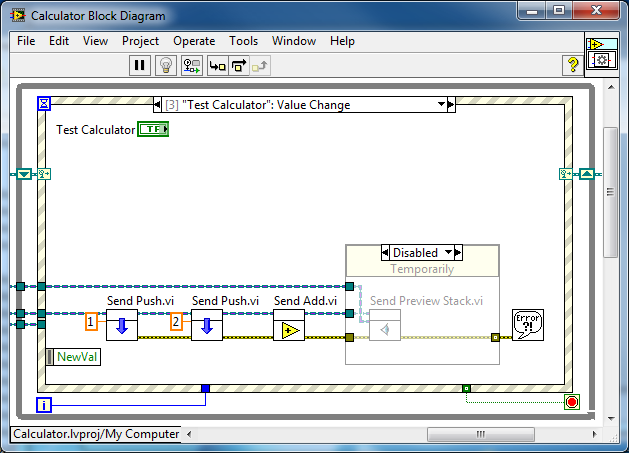
The project is almost finished. Why almost, only?

The *UPNCalculator* actor is used by *Calculator* actor and at the same time by the *UPNCalculatorGUI* actor. This can lead to race conditions, which means that the *Stack* of *UPNCalculator* contains unexpected values!

1. Start application again: *Calculator Launcher.lvlib: Splash Screen.vi*
   1. Execute a calculation using the *UPNCalculatorGUI*.
   2. Trigger a test calculation in the *Calculator* window.
   3. Update the *Stack Preview* in *UPNCalculatorGUI*

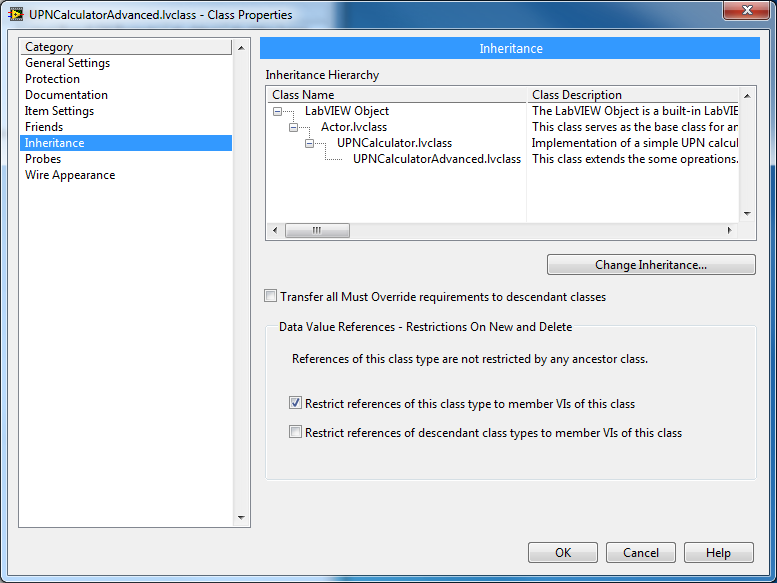
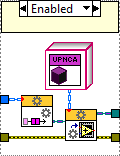
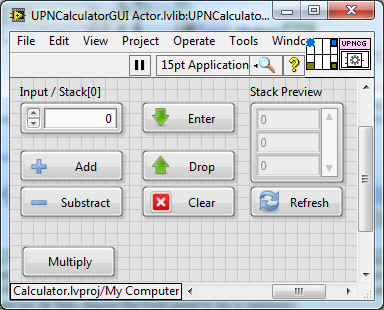
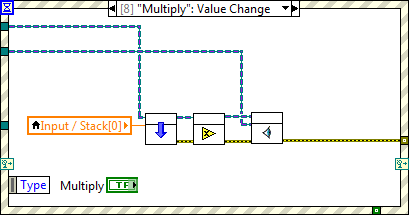
The *UPNCalculatorGUI* should use its own *UPNCalculator* actor.

1. Modify *UPNCalculatorGUI.lvclass:Actor Core.vi*: Start an additional *UPNCalculator* actor and store its *Enqueuer*  in the attributes of *UPNCalculatorGUI*.  
   
2. In *Stop Core.vi* send ist own *UPNCalculator* a *Send Normal Stop*.  
   
3. Disable periodic messages in *UPNCalculator.lvclass:Actor Core.vi*, to avoid typecast errors in the message handling of *UPNCalculatorGUI*!  
   
4. Start application again: *Calculator Launcher.lvlib: Splash Screen.vi*
   1. Execute a calculation using the *UPNCalculatorGUI*.
   2. Trigger a test calculation in the *Calculator* window.
   3. Update the *Stack Preview* in *UPNCalculatorGUI*
5. Now there should be no more *Stack* chaos.

In order to avoid unnecessary errors, disable *Send Preview Stack.vi* in *Calculator.lvlib:Calculator.lvclass:Actor Core.vi*.  


# Exercise 7: Advanced *UPNCalculator* (Inheritance)

Assume you had no access to *UPNCalculator Acotr.lvlib*, but you would like to add more operations to the *UPNCalculator*. You can create a new derived class extending *UPNCalculator*. Since you own the derived class you can feel free to add as many operations and corresponding messages you like[[3]](#footnote-3). You should use a similar structure in the project and on disk as the other actors.

1. Create a new library *UPCCalculatorAdvanced Actor.lvlib*.
2. Create a new class *UPCCalculatorAdvanced.lvlib*.
   1. Change inheritance to *UPNCalculator.lvlcass*. 
   2. Add static dispatch-Vis to implement more operations.
3. Use the *Actor Framework Message Makers* to generate the corresponding messages.
4. In *UPNCalculatorGUI.lvclass:Actor Core.vi* replace the object of *UPNCalculator* with an object of the new class.   
   
5. Add more buttons for the additional operations and corresponding event-cases.  
     
   
6. Start application: *Calculator Launcher.lvlib: Splash Screen.vi*
7. Test new operations.

# Notice

# Notice

# Notice

1. The queue reference is obtained at first access of *Stack* manipulation VIs automatically. [↑](#footnote-ref-1)
2. In the most simple case it could send the Stack to its Caller-Enqueuer. But this wouls implie a too strong coupling. [↑](#footnote-ref-2)
3. Extending the GUI is not so easy i fit is not prepared for this purpose. But, this challenge is left for an advanced course. [↑](#footnote-ref-3)