

Workshop

**Objekt-orientiertes Programmieren mit 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

Inhaltsverzeichnis

[UPN-Rechner 3](#_Toc345493776)

[Übung 1: Projekt aus Template erstellen 4](#_Toc345493777)

[Übung 2: Evaluieren der Applikation 8](#_Toc345493778)

[Übung 3: Erstellen der Klasse *Stack* 9](#_Toc345493779)

[Übung 4: Erweitern der Klasse *UPNCalculator Actor* 15](#_Toc345493780)

[Übung 5: Erweitern der Klasse *Calculator* 19](#_Toc345493781)

[Übung 6: Erweitern der Klasse *UPNCalculatorGUI Actor* 23](#_Toc345493782)

[Übung 7: Design Pattern Composition 28](#_Toc345493783)

[Übung 7: Advanced UPNCalculator (Vererbung) 30](#_Toc345493784)

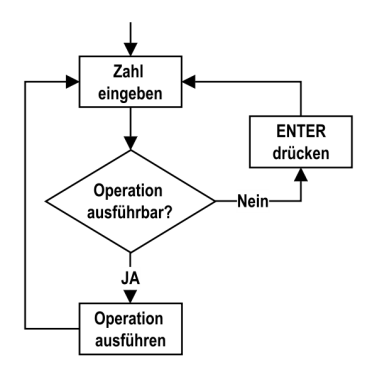
[Notizen 32](#_Toc345493785)

[Notizen 33](#_Toc345493786)

[Notizen 34](#_Toc345493787)

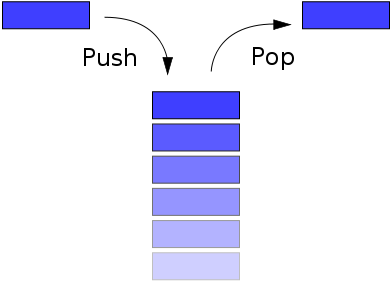
# UPN-Rechner

Ein UPN-Rechner (Umgekehte Polnische Notation) benutzt einen *Stack*, um die Operanden von Rechenoperationen zunächst auf den Stack zu legen. Bei der Ausführung einer Rechenoperation wird die notwendige Anzahl von Operanden vom Stack genommen, verarbeitet und das Ergebnis wieder auf den *Stack* gelegt.

[](http://upload.wikimedia.org/wikipedia/commons/e/ee/Upnlogik.svg)

Flussdiagramm der UPN (http://de.wikipedia.org/wiki/Umgekehrte\_Polnische\_Notation)

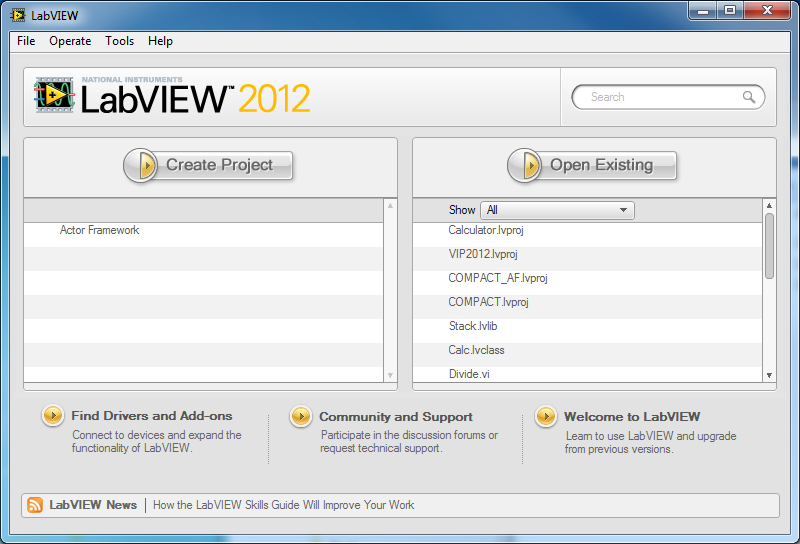
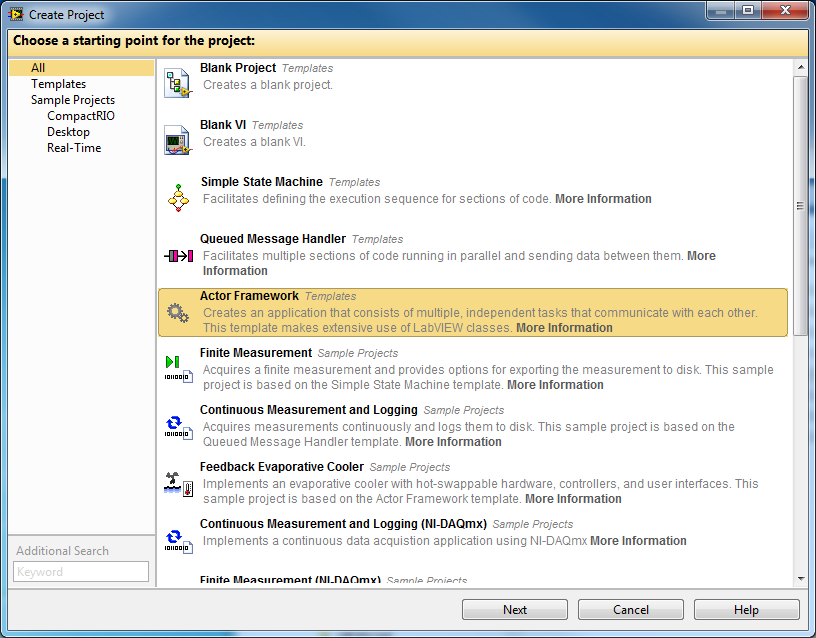
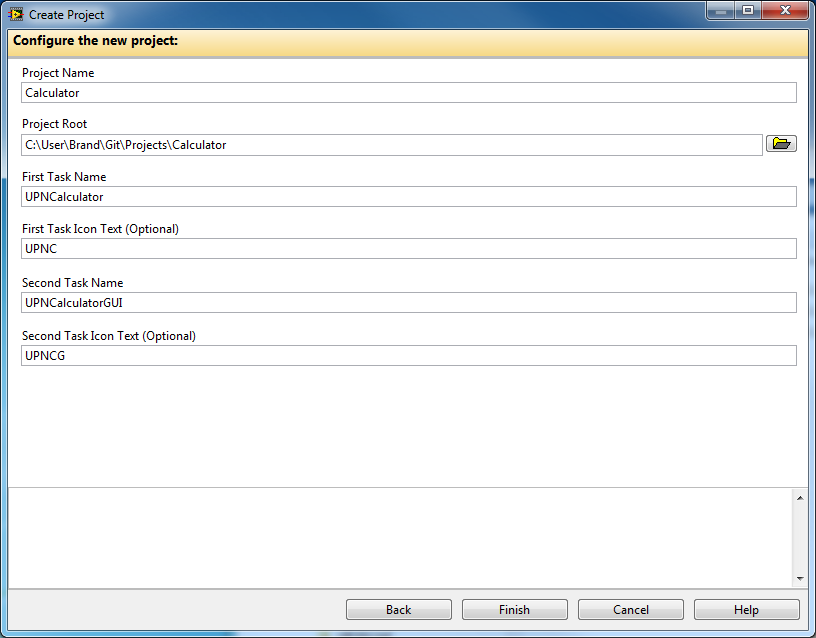
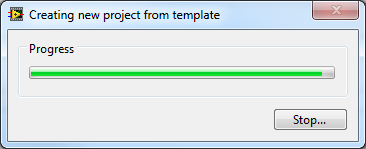
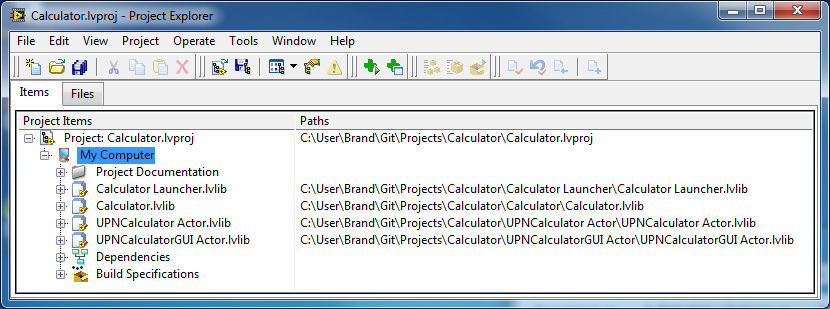
Ein Stack ist ein Stapelspeicher. Das Element, dass dem Stapel zuletzt mit *Push* hinzugefügt wurde, wird von dem Stapel mit *Pop* als erstes wieder entfernt. (lifo – last in, first out).

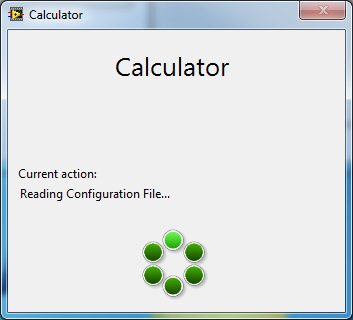
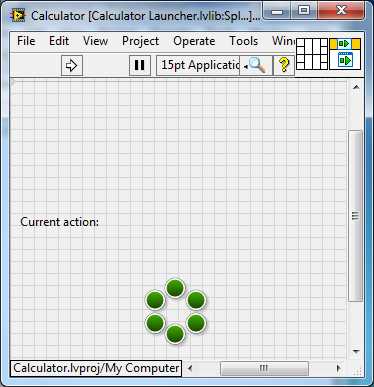
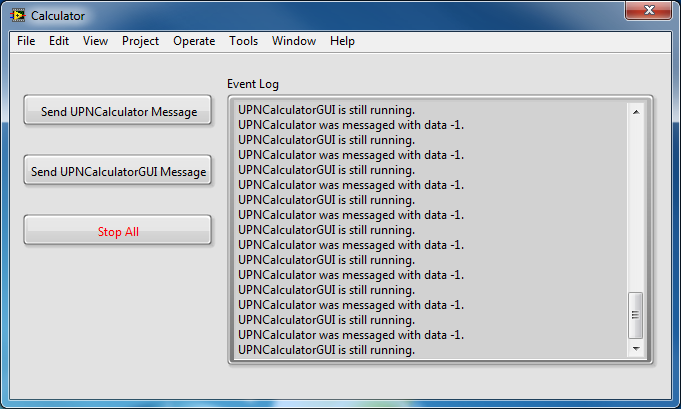
[](http://upload.wikimedia.org/wikipedia/commons/2/29/Data_stack.svg)

Schema eines Stapelspeichers (http://de.wikipedia.org/wiki/Stapelspeicher)

# Übung 1: Projekt aus Template erstellen

In Übung 1 lernen Sie, wie man mit dem Project Wizard ein neues *Actor Framework*-Projekt erstellt. Wählen Sie im LabVIEW Startbildschirm *Create Project* und selektieren Sie die *Actor Framework* Vorlage. Ein Assistent führt Sie durch die einzelnen Konfigurationsschritte. So werden automatisch verschiedene Bibliotheken erstellt, z. B. der *Splash Screen*, der die eigentliche Applikation in Form eines *Actors* startet. Nach dem Durchlaufen der Schritte wird aus den Angaben ein bereits lauffähiges LabVIEW-Projekt erzeugt, das die Basis für die eigentliche Anwendung bildet.

* 1. Starten Sie LabVIEW.
  2. Erstellen Sie ein neues Projekt (**Create Project**). 
* Wählen Sie im LabVIEW Startbildschirm *Create Project*, selektieren Sie die *Actor Framework* Vorlage und klicken Sie *Next*.  
  
* Geben Sie als Projektname **Calculator** ein. Wählen Sie ein geeignetes Verzeichnis, füllen Sie die weiteren Felder wie angegeben aus und klicken Sie auf *Finish*.  
  
* Warten Sie ab bis das Projekt erstellt wurde.  
  
* Das *Calculator* Projekt sollte nun so aussehen.  
  
* Projektstruktur:
  1. Die Projekt Dokumentation enthält zwei wichtige Dokumente:
     + *Actor Framework Whitepaper.html* enthält die Konzepte und Hintergrundwissen
     + *Actor Framework.html* enthält Informationen über dieses Template.
  2. *Calculator Launcher.lvlib* enthält das *Splash Screen.vi*, dass die eigentliche Applikation als *Actor* startet.
     + *Localization.vi* dient der Übersetzung von Texten in verschiedene Sprachen. Es ist in allen vorbereiteten *Actor*-Klassen enthalten. Es wird in dieser Übung jedoch nicht weiter betrachtet.
  3. Die anderen LabVIEW Bibliotheken enthalten die Actor-Klassen und die zugehörigen Nachrichten, die der Wizzard für dieses Projekt erstellt hat.
     + *Calculator.lvlib* enhält die *Calculator.lvclass*. Eine Instanz dieser Klasse wir von dem *Splash Screen.vi* gestartet. Sie ist die eigentliche Applikation. Sie startet zwei *Actors*:
       - *UPNCalculator Actor.lvlib:UPNCalculator.lvclass*  
         In dieser Klasse wir in der Übung der UPN-Rechner implementiert. Er sendet sich selbst periodisch Nachrichten und hat keine GUI vorbereitet.
       - *UPNCalculatorGUI Actor.lvlib:UPNCalculatorGUI.lvclass*  
         In dieser Klasse wird das GUI für den UPN-Rechner implementiert. Es beinhaltet bereits eine Eventschleife.
     + Starten der Applikation

1. Erweitern Sie *Calculator Launcher.lvlib* durch klicken auf das vorangestellte +.
2. Öffnen Sie das *Splash Screen.vi* und klicken Sie auf den *Run-Button*. Die Klasse *Calculator.lvclass* wird geladen und gestartet.  
    
3. Der Fenster des Calculator-Actors sieht so aus:  
   
   1. Zwei weitere Aktoren werden im Hintergrund gestartet, *UPNCalculator* und *UPNCalculatorGUI*. Beide senden in regelmäßigen Abständen Nachrichten als Lebenszeichen an den *Calculator* Aktor.
   2. Klicken Sie auf Send *UPNCalculator Message* und *UPNCalculatorGUI* Message, um Nachrichten an die beiden Aktoren zu senden. Der jeweils adressierte Aktor sendet asynchron eine Nachricht an den Calculator zurück, die von diesem in das *Event Log* geschrieben wird.
   3. Klicken Sie *Stop All*, um die Applikation zu beenden.

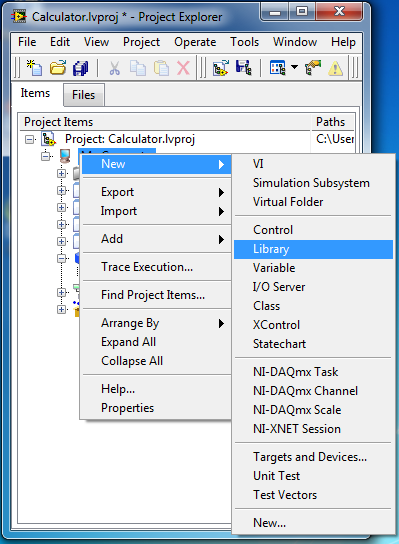
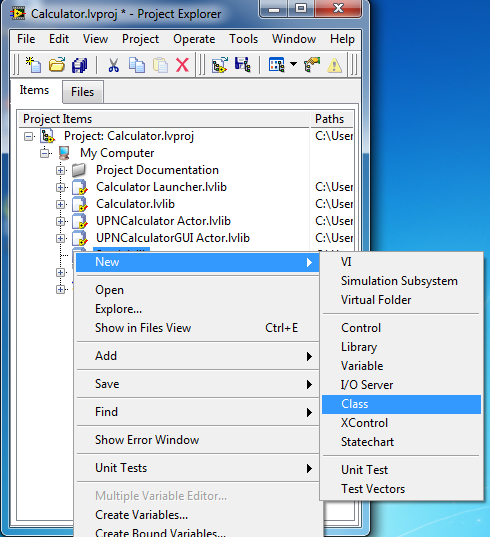
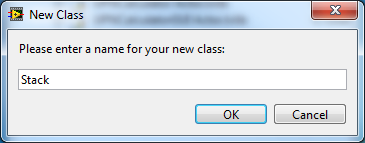
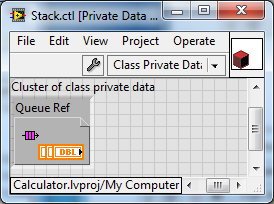
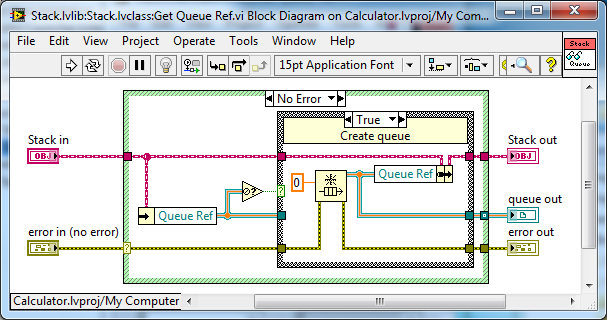
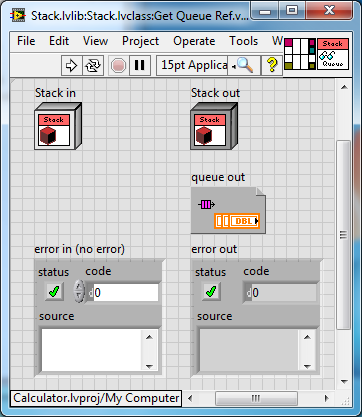
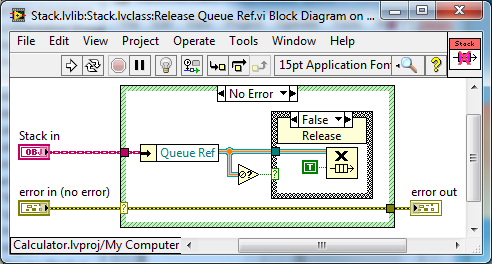
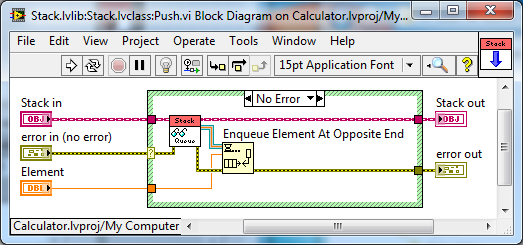
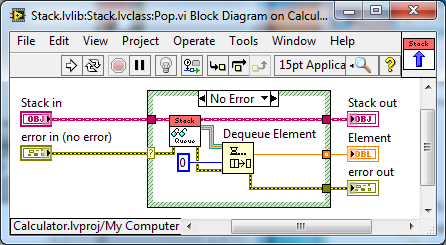
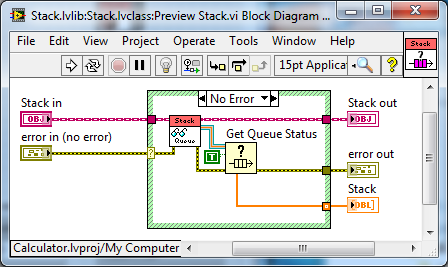
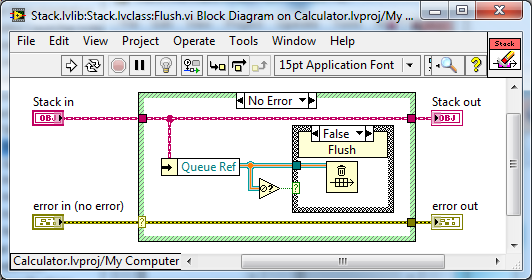
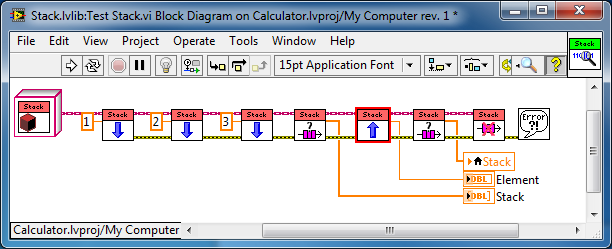
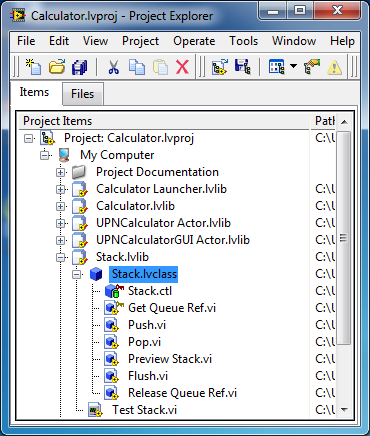
# Übung 2: Evaluieren der Applikation

In Übung 2 sehen wir uns die Applikation näher an.

* An dieser Stelle zeige ich wie das Actor Framework-Template funktioniert und an welchen Stellen der Anwender eingreifen muss, um es zu erweitern.

# Übung 3: Erstellen der Klasse *Stack*

In der folgenden Übung soll eine Queue benutzt werden, um den *Stack* zu implementieren.

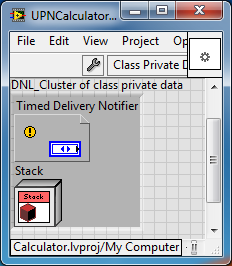
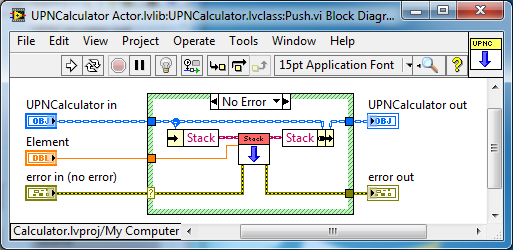
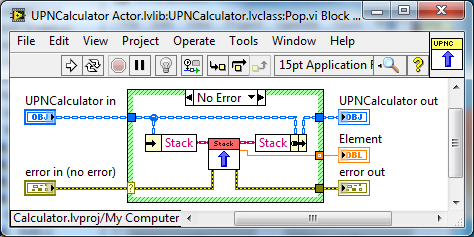
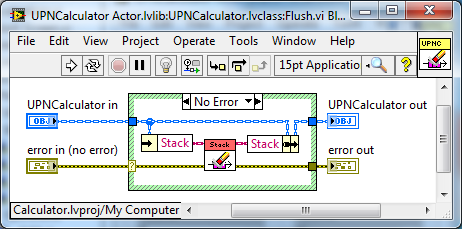
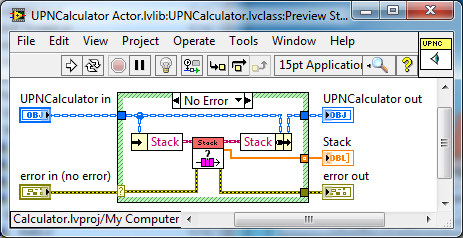
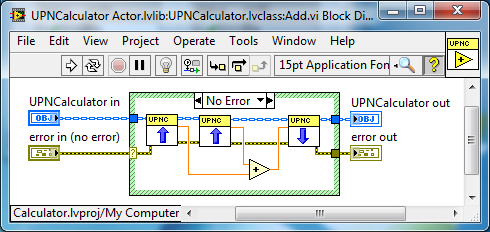
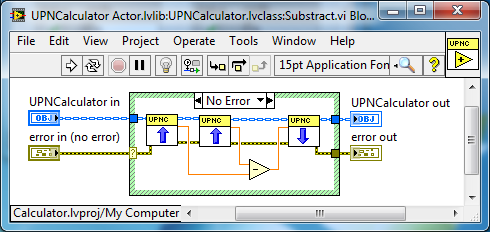
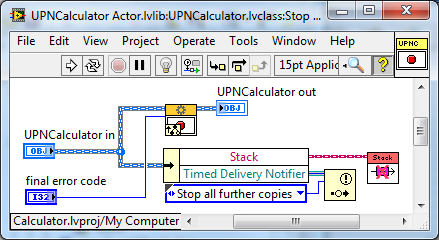
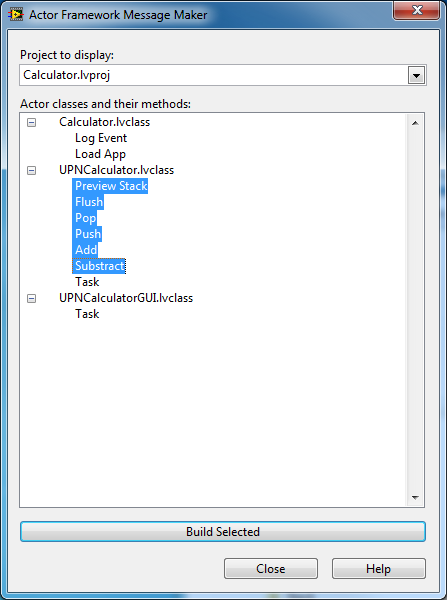
* Erstellen Sie eine neue Bibliothek *Stack*:  
  
* Rechtsklick auf die neue Bibliothek *Save->Save All (this Library)*  
  Wählen Sie ein neues Verzeichnis in Ihrem Projektverzeichnisbaum.
* Wählen Sie im Kontextmenü der Bibliothek *Stack.lvlib* Properties aus.
  + In der Kategorie *General Settings* editieren Sie das Icon. Setzen Sie *Stack* in Line 1 des Icon Text.
  + In der Kategorie *Documentation* geben Sie eine kurze Beschreibung ein.
* Neue Klasse *Stack* erstellen: Rechtsklick auf *Stack.lvlib->New->Class*  
  
* Rechtsklick auf die neue Klasse Save->Save All (this Class)  
  Wählen Sie das neues Verzeichnis in Ihrem Projektverzeichnisbaum.
* Wählen Sie im Kontextmenü der Bibliothek *Stack.lvclass* Properties aus.
  + In der Kategorie *General Settings* editieren Sie das Icon. Setzen Sie *Stack* in Line 1 des Icon Text.
  + In der Kategorie *Documentation* geben Sie eine kurze Beschreibung ein.
* Öffnen Sie das Attribut Control *Stack.ctl* und fügen eine *Queue*-Referenz für den Datentype *Double* ein. Danach schließen Sie die Control.  
  
* Erstellen Sie ein neues *static dispatch*-VI über das Kontextmenü der Klasse und speichern Sie es mit dem Namen *Get Queue Ref.vi*.  
    
  Dieses VI soll die Referenz auf die in den Attributen gespeicherte *Queue* zurückgeben. Falls die Referenz noch nicht gültig ist, wird die *Queue* erzeugt und in den Attributen für künftige Zugriffe gespeichert. **Setzen Sie den *Access Scope* auf *Protected*!**
* Erstellen Sie ein neues *static dispatch*-VI über das Kontextmenü der Klasse und speichern Sie es mit dem Namen *Release Queue.vi*.  
  
* Erstellen Sie ein neues *static dispatch*-VI über das Kontextmenü der Klasse und speichern Sie es mit dem Namen *Push.vi*.  
  
* Erstellen Sie ein neues *static dispatch*-VI über das Kontextmenü der Klasse und speichern Sie es mit dem Namen *Pop.vi*.  
  
* Erstellen Sie ein neues *static dispatch*-VI über das Kontextmenü der Klasse und speichern Sie es mit dem Namen *Preview Stack.vi*.  
  
* Erstellen Sie ein neues static dispatch-VI über das Kontextmenü der Klasse und speichern Sie es mit dem Namen *Flush.vi*. In diesem VI wird auf das *Get Queue Ref.vi* verzichtet, weil nicht unnötig eine Queue erzeugt werden soll.  
  
* Erstellen Sie ein neues VI über das Kontextmenü der Bibliothek und speichern Sie es mit dem Namen *Test Stack.vi*. Probieren Sie die Funktionalität des *Stack* aus.  
  
* Die *Stack*-Klasse sollte jetzt ungefähr so aussehen.  
  

# Übung 4: Erweitern der Klasse *UPNCalculator Actor*

In Übung 4 erweitern Sie die *UPNCalculator* Klasse um den *Stack* und fügen einfache Rechenoperationen hinzu. Mit Hilfe des *Actor Framework Message Maker* erzeugen Sie die zugehörigen Nachrichten.

Wir benutzen hier das **Delegation Pattern:**

Die Klasse *Stack* ist wiederverwendbar. Die *UPNCalculator*-Klasse delegiert die Stack-Manipulationen an diese Klasse. Sie bleibt nach außen für den Anwender der *UPNCalculator*-Klasse unsichtbar.

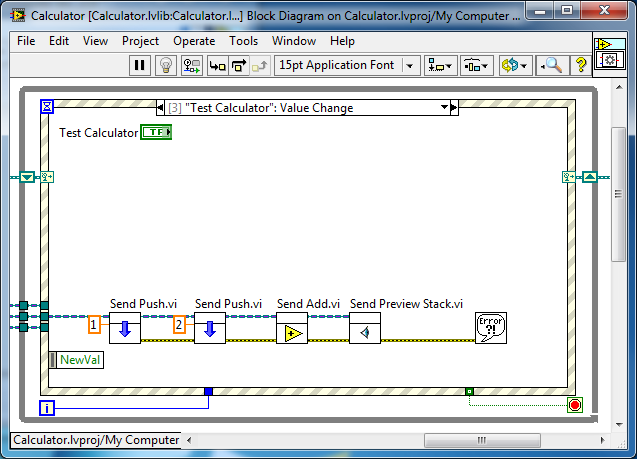
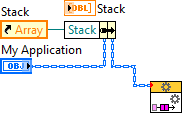
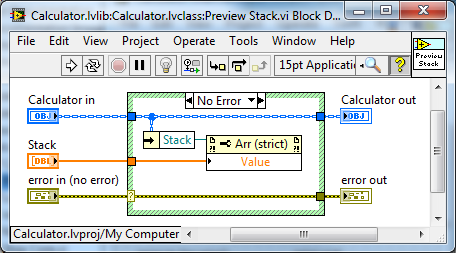
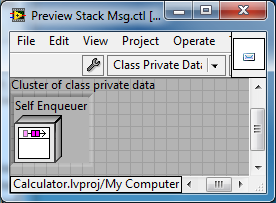
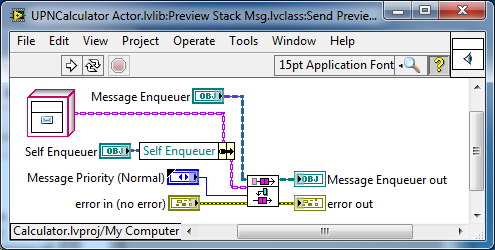
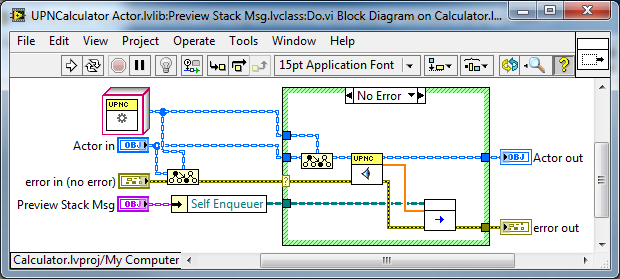
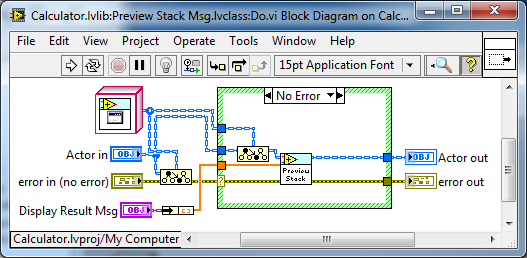
* Öffnen Sie *UPNCalculator.ctl*. Fügen Sie ein Objekt der Klasse *Stack* zu den Attributen hinzu und schließen Sie das *Control*.  
  
* Fügen Sie *Wrapper*-VIs (*static dispatch*) für die *Stack*-Methoden, *Push*, *Pop*, *Flush* und *Preview Stack* hinzu. Legen Sie dazu einen *Virtual Folder* in der *UPNCalculator*-Klasse an.   
    
    
    
  
* Legen Sie in einem neuen virtuellen Ordner *Operations* für die *static disptach*-VIs für die Operationen Addieren und Subtrahieren an.  
    
  
* Selbstverständlich können Sie bei Bedarf weitere Operationen hinzufügen oder in einer Kindklasse implementieren.
* Modifizieren Sie das *Stop Core.vi*, um die *Queue* freizugeben.  
  
* Benutzen Sie *Tools->Actor Framework Message Maker…* um die korrespondierenden Nachrichten zu erzeugen.  
    
  Wählen Sie die markierten VIs aus und klicken Sie *Build Selected*. Danach schließen Sie den *Message Maker*.
* Speichern und schließen Sie alle erzeugten VIs.
* Verschieben Sie die neuen *Message*-Klassen in den virtuellen Ordner: *Messages for UPNCalculator*. Objekte dieser Nachrichten-Klassen werden später vom GUI an den Rechner gesendet.

Die *UPNCalculator*-Klasse ist nun vorbereitet, Nachrichten zu empfangen, um Zahlen auf den *Stack* zu legen und mit diesen Rechenoperationen auszuführen.

Die Nachricht *Preview Stack Msg.lvclass* muss allerdings später noch bearbeitet werden, damit das Ergebnis einer Rechenoperation zurück zum GUI zur Anzeige gelangt. Dazu muss aber erst die entsprechende Nachricht in der *UPNCalculatorGUI*-Klasse erstellt werden.

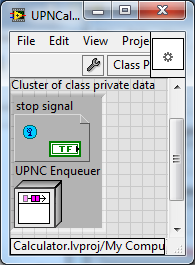
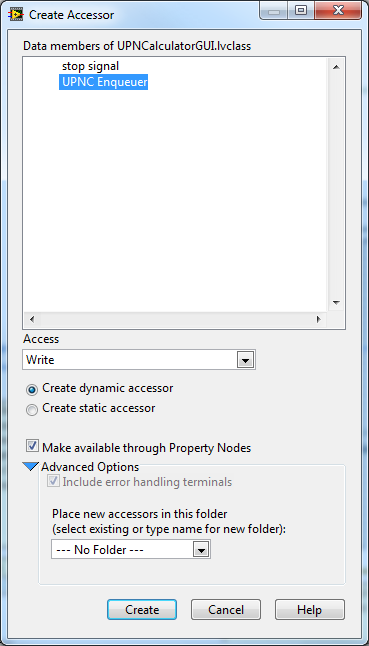
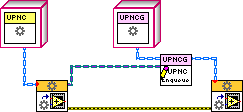
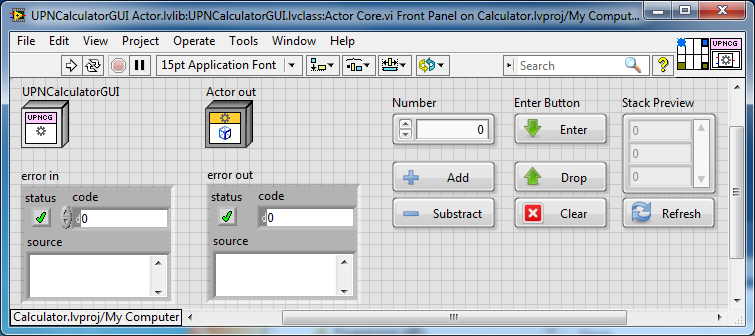
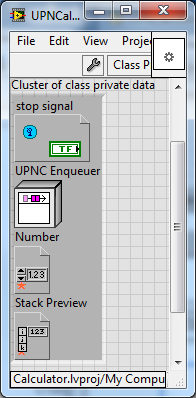
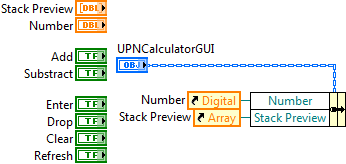
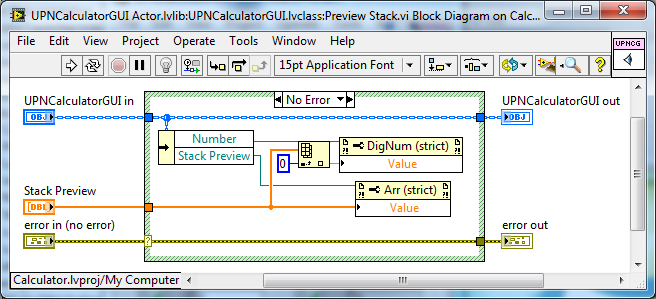
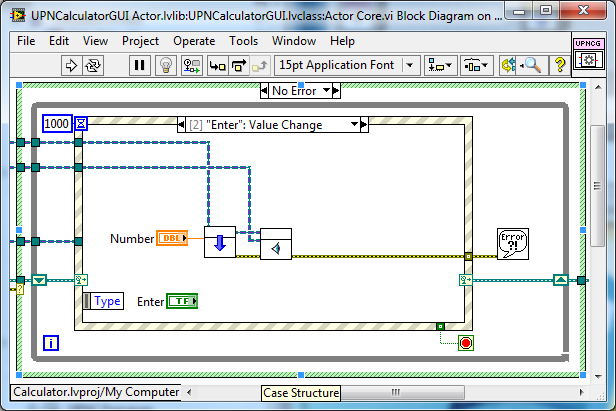
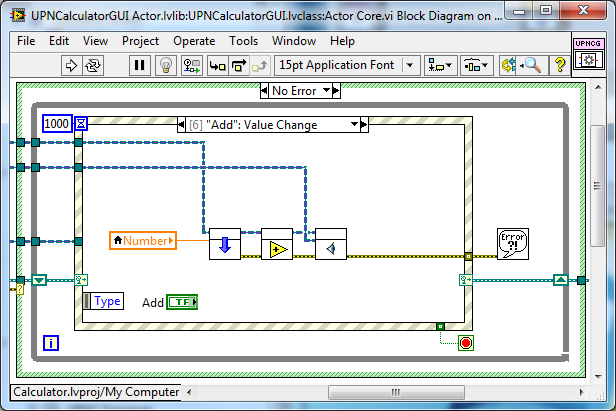
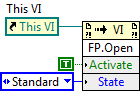
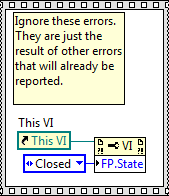
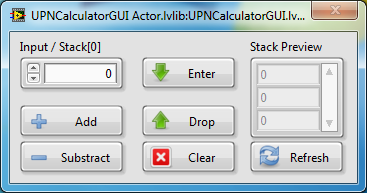
# Übung 5: Erweitern der Klasse *Calculator*

Um in einem ersten Schritt die Klasse *UPNCalculator* für Rechnungen zu benutzen, fügen wir einen zusätzlichen Event-Case zur *Calculator*-Klasse hinzu.

* Öffnen Sie das Calculator.lvclass:Actor Core.vi.  
  In diesem VI werden u. a. die Aktoren UPNCalculator und UPNCalculatorGUI gestartet bevor *Call Parent Methode*, das VI, das die Nachrichten empfängt, aufgerufen wird. Parallel wir eine Ereignisschleife gestartet, in der programmatische Ereignisse und Benutzereingaben behandelt werden.
* Platzieren Sie einen weiteren Button auf das Frontpanel und beschriften Sie ihn mit *Test Calculator*.
* Erzeugen Sie für diesen Button einen neuen *Event-Case*.
* Fügen Sie die folgenden VIs in diesen *Case* ein:
  + *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Push.vi*
  + *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Push.vi*
  + *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Add.vi*
  + *UPNCalculator Actor.lvlib:Push Msg.lvclass:Send Preview Stack.vi*
  + *Simple Error Handler.vi*
* Des resultierende *Event Case* sollte etwas so aussehen:  
  
* Speichern Sie die Änderung und schließen Sie das VI.
  + - Starten der Applikation: *Calculator Launcher.lvlib: Splash Screen.vi*
    - Lösen Sie eine Berechnung durch Klicken auf den neuen Button aus. Um das Ergebnis anzusehen sollten Sie das VI *UPNCalculator Actor.lvlib:UPNCalculator.lvclass:Preview Stack.vi* öffnen.
    - Fügen Sie auf dem Frontpanel noch ein *Double-Array* als Indikator hinzu, um den *Stack* anzuzeigen.
    - Erzeugen Sie ein Referenz-*Control* des *Stack*-Indikators und fügen sie diesen den Attributen hinzu. Schreiben Sie die Referenz in das Objekt, bevor irgendetwas anderes mit ihm gemacht wird.   
      
    - Erstellen Sie ein *static dispatch*-VI, um den Indikator via *Property-Node* zu setzen. Speichern Sie es als *Preview Stack.vi*.  
      
    - Erzeugen Sie die zugehörige Nachricht mit dem *Actor Framework Message Maker*.
    - Fügen Sie dem Attribut der *Preview Stack Msg.lvclass* in *UPNCalculator Actor.lvlib* ein *Self Enqueuer* Objekt hinzu. (Erzeugen Sie eine Kontrolle des Ausgangs vom VI *Read Self Enqueuer.vi* in der AF-Funktionspalette.)  
      
    - Das Enqueuer-Objekt ist notwendig, weil das *UPNCalculator.lvclass:Preview Stack.vi* den Stack nur zurückgibt, aber sonst nicht weis und auch nicht wissen soll[[1]](#footnote-1), was damit zu tun ist. Das Message-Objekt ist dafür zuständig. Der Message-Dispatcher muss wissen was mit dem Stack zu tun ist. Im einfachen Fall wird der Stack an den Message-Dispatcher zurück gesendet. Er muss dafür eine entsprechende Methode bereit stellen. Der Stack könnte aber auch an einen anderen Actor gesendet werden.
    - Modifizieren Sie *Preview Stack Msg.lvclass:Send Preview Stack.vi*, um den *Self Enqueuer* zu setzen.   
        
      **Setzen Sie den *Self Enqueuer* auf der *Connector-Pane* auf *requiered*!**
    - Modifizieren Sie *Preview Stack Msg.lvclass:Do.vi*, um den *Stack*-Inhalt mittels *Calculator.lvlib:Preview Stack Msg.lvclass:Send Preview Stack.vi* an den *Self Enqueuer zurück zu senden*.  
      
    - Modifizieren Sie *Calculator.lvlib:Preview Stack Msg.lvclass:Do.vi*, um den *Stack*-Inhalt auch wirklich anzuzeigen.  
      
    - Starten Sie die Applikation erneut: *Calculator Launcher.lvlib: Splash Screen.vi*
    - Beobachten Sie das Ergebnis nach dem Auslösen der Berechnung.

# Übung 6: Erweitern der Klasse *UPNCalculatorGUI Actor*

In Übung 4 erweitern Sie die *UPNCalculatorGUI* Klasse um die Referenz auf den *UPNCalculator ( Design Pattern: Aggregation)*, die Zahleneingabe, bzw. –anzeige und fügen die Knöpfe hinzu, die das Senden von entsprechenden Nachrichten an den *UPNCalculator* auslösen. Sie fügen Methoden bzw. Nachrichten hinzu, die den Empfang und die Darstellung des Stack-Inhalts erlauben.

* Erweitern Sie das Attribut der *UPNCalculatorGUI*-Klasse um den Enqueuer des UPNCalculator. Dieses Objekt wird benutzt, um dem *UPNCalculator Actor* Nachrichten zu senden.  
  
* Erzeugen Sie mittel Kontextmenü der Klasse ein neues *Data Member Access VI* für das Attribut *UPNC Enqueuer*.   
  
* Setzen Sie den *UPNC Enqueuer* in *Calculator.lvlib:Calculator.lvclass:Actor Core.vi*.  
  
* Öffnen Sie nun das *UPNCalculator.lvclass:Actor Core.vi*. Platzieren Sie auf dem Frontpanel die Bedienelemente für den Taschenrechner:  
  
  + Numerisches Control für die Zahleneingabe
  + Buttons für die Stackmanipulation:
    - Enter (Push an element to Stack)
    - Drop (Pop an element from Stack)
    - Clear (Flush Stack)
    - Refresh (Update Stack Preview)
  + Buttons für die Rechenoperationen:
    - Add
    - Substract
* Fügen Sie Referenzen auf die Elemente *Number* und *Stack Preview* in die Attribute ein und setzen Sie diese im Blockdiagram bevor irgendetwas anderes mit der Klasse getan wird.  
   
* Erstellen Sie ein *static dispatch*-VI, um den *Stack*-Inhalt via *Property-Node* zu setzen. Speichern Sie es als *Preview Stack.vi*.  
  
* Erzeugen Sie die korrespondierende Nachricht mit Hilfe des *Actor Framework Message Maker*.
* Ersetzen Sie in *UPNCalculator Actor.lvlib:Preview Stack Msg.lvclass:Do.vi* das *Calculator.lvlib:Preview Stack Msg.lvclass:Send Preview Stack.vi* mit *UPNCalculatorGUI Actor.lvlib:Preview Stack Msg.lvclass:Send Preview Stack.vi*.
* Öffnen Sie nun das *UPNCalculator.lvclass:Actor Core.vi*. Fügen Sie für die neuen Knöpfe die entsprechenden Event-Cases hinzu und versenden darin die notwendigen Nachrichten an den UPNCalculator.  
  + **Enter:**  
    
  + **Add:** etc.
* Vergessen Sie nicht das Fronpanel prgrammatisch zu öffnen und zu schließen:  
  Am Beginn:  Am Ende: 
* Stellen Sie die Window Apparence nach Ihren Wünschen ein.
* Starten Sie die Applikation erneut: *Calculator Launcher.lvlib: Splash Screen.vi*
* *Benutzen Sie das UPNCalculatorGUI zur Berechnung.*

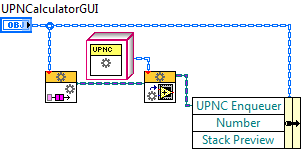
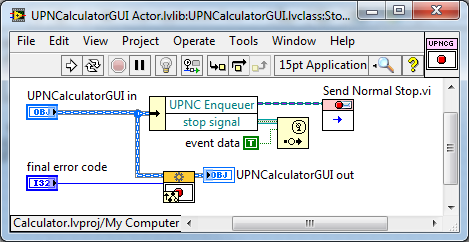
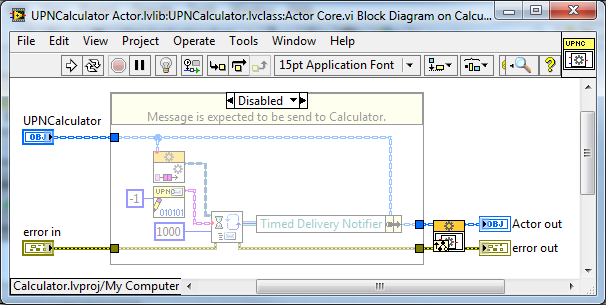
# Übung 7: Design Pattern Composition

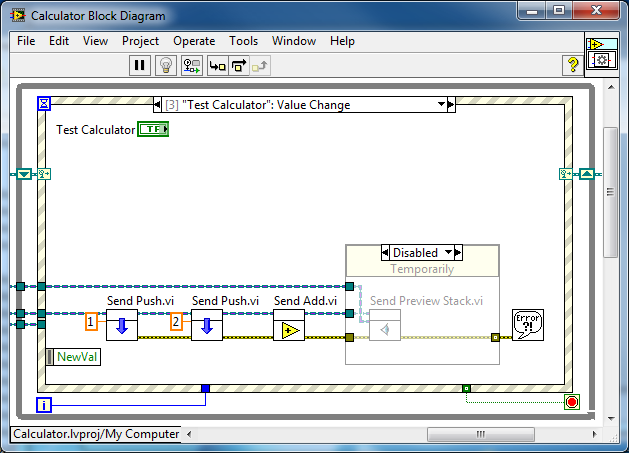
Damit ist das Projekt fast abgeschlossen. Warum nur fast?

Der *UPNCalculator*-Aktor wird sowohl von dem *Calculator*-Aktor als auch von dem *UPNCalculatorGUI*-Aktor benutzt. Das kann dazu führen, dass der *Stack* des *UPNCalculator* in einen unerwarteten Zustand gerät.

* Starten Sie die Applikation erneut: *Calculator Launcher.lvlib: Splash Screen.vi*
  + Führen Sie eine Berechnung mit dem *UPNCalculatorGUI* aus.
  + Triggern Sie die Testberechnung im *Calculator*-Fenster.
  + Aktualisieren Sie den *Stack Preview* im *UPNCalculatorGUI*

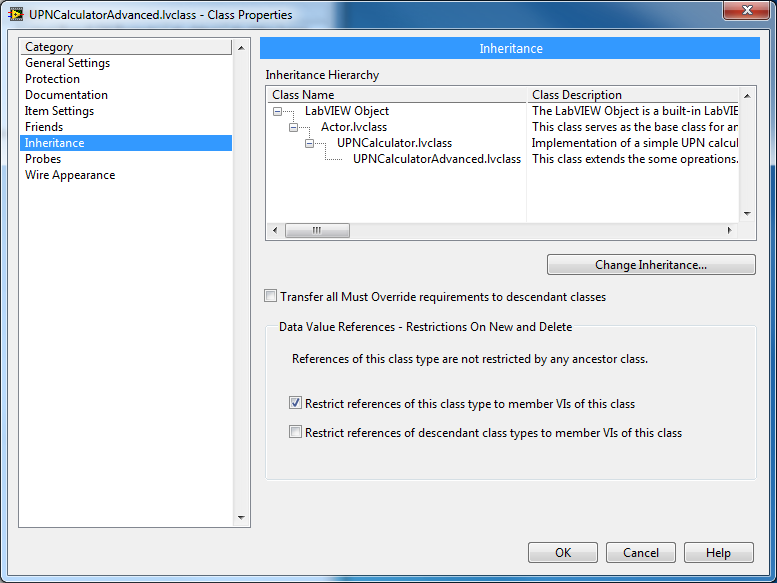
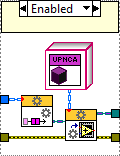
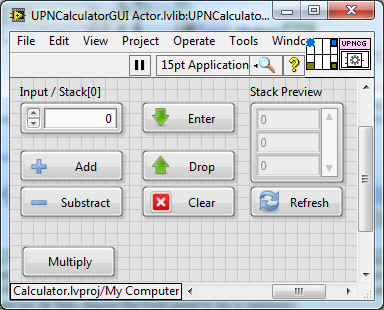
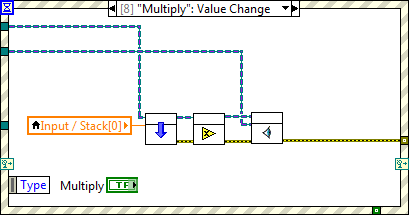
Das *UPNCalculatorGUI* sollte also seinen eigenen *UPNCalculator*-Aktor benutzen.

* Modifizieren Sie UPNCalculatorGUI.lvclass:Actor Core.vi: Starten Sie einen eigenen UPNCalculator-Aktor und merken sich seien Enqueuer in den Attributen.  
  
* Im *Stop Core.vi* senden sie ihrem eigenen *UPNCalculator* ein *Send Normal Stop*.  
  
* Deaktivieren Sie die periodischen Nachrichten in *UPNCalculator.lvclass:Actor Core.vi*, um Typecast-Fehler in der Nachrichtenbehandlung des *UPNCalculatorGUI* zu vermeiden!  
  
* Starten Sie die Applikation erneut: *Calculator Launcher.lvlib: Splash Screen.vi*
  + Führen Sie eine Berechnung mit dem *UPNCalculatorGUI* aus.
  + Triggern Sie die Testberechnung im *Calculator*-Fenster.
  + Aktualisieren Sie den *Stack Preview* im *UPNCalculatorGUI*
* Jetzt sollte die Stack-Verwirrung nicht mehr zu beobachten sein.

Um unnötige Fehler zu vermeiden, kommentieren Sie das *Send Preview Stack.vi* in dem *Calculator.lvlib:Calculator.lvclass:Actor Core.vi* aus.  


# Übung 7: Advanced UPNCalculator (Vererbung)

Nehmen Sie an Sie hätten keinen Zugriff auf die *UPNCalculator Acotr.lvlib*. Sie möchten den Rechner aber dennoch um weitere Operationen erweitern. Sie erzeugen einfach eine neue Klasse und lassen diese von dem *UPNCalculator* erben und fügen in dieser Klasse Operationen hinzu und generieren die zugehörigen Nachrichten[[2]](#footnote-2). Bei Speichern orientieren Sie sich an der Verzeichnisstruktur der anderen Bibliotheken und Klassen in diesem Projekt.

* Erzeugen Sie eine neue Bibliothek *UPCCalculatorAdvanced Actor.lvlib*.
* Erzeugen Sie eine neue Klasse *UPCCalculatorAdvanced.lvlib*.
  + Lassen Sie die Klasse von *UPNCalculator.lvlcass* erben. 
* Fügen Sie *static dispatch*-VIs hinzu, die weitere Rechenoperationen implementieren.
* Mit Hilfe des *Actor Framework Message Makers* erzeugen Sie die zugehörigen Nachrichten.
* In *UPNCalculatorGUI.lvclass:Actor Core.vi* ersetzen Sie das Objekt des *UPNCalculator* mit einem Objekt der neuen Klasse.   
  
* Fügen weitere Buttons für die neuen Rechenoperationen mit zugehörigen Event-Cases ein.  
    
  
* Starten Sie die Applikation erneut: *Calculator Launcher.lvlib: Splash Screen.vi*
* Probieren Sie die neuen Rechenoperationen aus.

# Notizen

# Notizen

# Notizen

1. Im einfachsten Fall könnte es den Stack an den Caller-Enqueuer senden. Das würde aber eine zu starke Kopplung bedeuten. [↑](#footnote-ref-1)
2. Die GUI-Klasse zur Erweitern ist nicht so einfach, wenn Sie dafür nicht vorbereitet wurde. Diese Herausforderung bleibt für einen Fortgeschrittenenkurs. [↑](#footnote-ref-2)