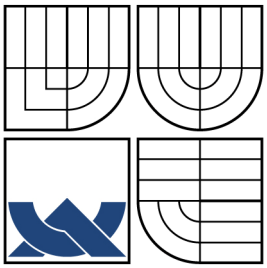
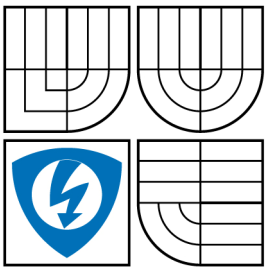


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OFCONTROL AND INSTRUMENTATION

OBJEKTOVÉ PROGRAMOVÁNÍ V LABVIEW 8.5

OBJECT PROGRAMMING IN LABVIEW 8.5

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JOSEF ROHÁČ
AUTHOR
VEDOUCÍ PRÁCE — Ing. MILOSLAV ČEJKA CSc.
SUPERVISOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Josef Roháč	***ID:***	78416
Ročník:	3	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Objektové programování v LabVIEW 8.5

POKYNY PRO VYPRACOVÁNÍ:

1/Seznamte se s objektovým programováním v LabVIEW 8.5,
2/Porovnejte tento přístup s klasickým dataflow programováním v LabVIEW
3/Vypracujte dvě úlohy do počítačových cvičení, které dokumentují vlastnosti objektového programování v LabVIEW
4/Navržené úlohy ověřte a zhodnoťe

DOPORUČENÁ LITERATURA:

Firemní literatura National Instruments (www.ni.com)

Termín zadání:	9.2.2009	***Termín odevzdání:***	1.6.2009

Vedoucí práce: Ing. Miloslav Čejka, CSc.

prof. Ing. Pavel Jura, CSc.
předseda oborové rady

Abstrakt

Cílem této práce je objasnit a seznámit uživatele s problematikou objektového programování v LabVIEW verze 8.5. V úvodu nastíní princip LabVIEW jako takového a připomene, kdy bylo objektové programování poprvé implementováno v tomto vývojovém prostředí. Dále postupně rozebírá teorii tříd a objektů, jejich vytváření, základní vlastnosti, použití v blokovém diagramu a různá nastavení těchto objektů potřebné k jejich správnému a plnému využití při programování. Teoretická část končí studií hierarchie dědění mezi objekty a ještě se zabývá dynamickými a statickými vstupy a výstupy.

Pro představu o rozdílech mezi objektovým programováním v LabVIEW a v C++ je zařazeno srovnání mechanismů a principů teoreticky i názorně na ukázkové úloze. Rovněž srovnává klasický dataflow přístup s objektovým programováním v LabVIEW. A na závěr jsou přiloženy dvě vypracované úlohy do cvičení včetně návodů.

Klíčová slova

Objektové programování, LabVIEW, dataflow

Abstract

The aim of this work is to clarify and to familiarize users with the problem of object programming in LabVIEW version 8.5. In the introduction outlining the principle of LabVIEW as such a recall when it was first implemented programming object in the development environment. Furthermore, gradually examines the theory of classes and objects, their creation, basic properties, use in the block diagram and the various settings of the objects required for their proper and full use in the programming. The theoretical part of the end of the study the inheritance hierarchy between objects, and even deals with dynamic and static inputs and outputs.

For the idea of the differences between the object programming in LabVIEW and C++ is included comparison of mechanisms and principles theoretically and clearly on the example programme. It also compares classical dataflow approach and object programming in LabVIEW. And finally they are accompanied by two developed by the task, including manual.

Keywords

Object programming, LabVIEW, dataflow

Bibliografická citace

ROHÁČ, J. Objektové programování v LabVIEW 8.5. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 42 s. Vedoucí bakalářské práce Ing. Miloslav Čejka, CSc.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Objektové programování v LabVIEW8.5 jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: **1. června 2009** ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Miloslavu Čejkovi, CSc. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **1. června 2009** ……

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

1. ÚVOD

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) se zakládá na použití tzv. virtuálních přístrojů a jejich programové implementace. Takto realizovaný meřicí přístroj má tři části:

- ovládací panel, sloužící k prezentaci výsledků měření a spouštění měření obsluhou

- stykovou jednotku, která zajišťuje kontakt s měřicím systémem (např. na počítač), přes kterou se posílají výsledky měření dále do meřicího obvodu

- vlastní měřicí část, provádějící měření (např. A/D převodník) a zpracovává výsledky

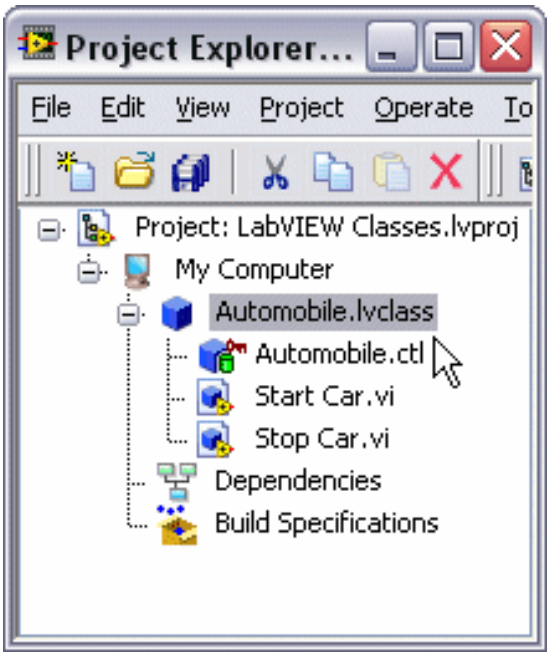
Virtuální přístroj pak běží na počítači, na jehož monitoru je zobrazen ovládací panel.

První verze LabVIEW (2.5.2) pro počítače IBM – PC byla uvedena na trh firmou National Instruments v roce 1992. Následovaly další verze, z nichž každá přinášela řadu různých vylepšení, nové funkce, lepší kompatibilitu a v neposlední řadě i objektově orientované programování, které se poprvé objevilo ve verzi 8.2 (r. 2006). V dnešní době máme k dispozici, nedávno vydanou, nejnovější verzi 8.6 (r. 2008).

2. TŘÍDY A OBJEKTY

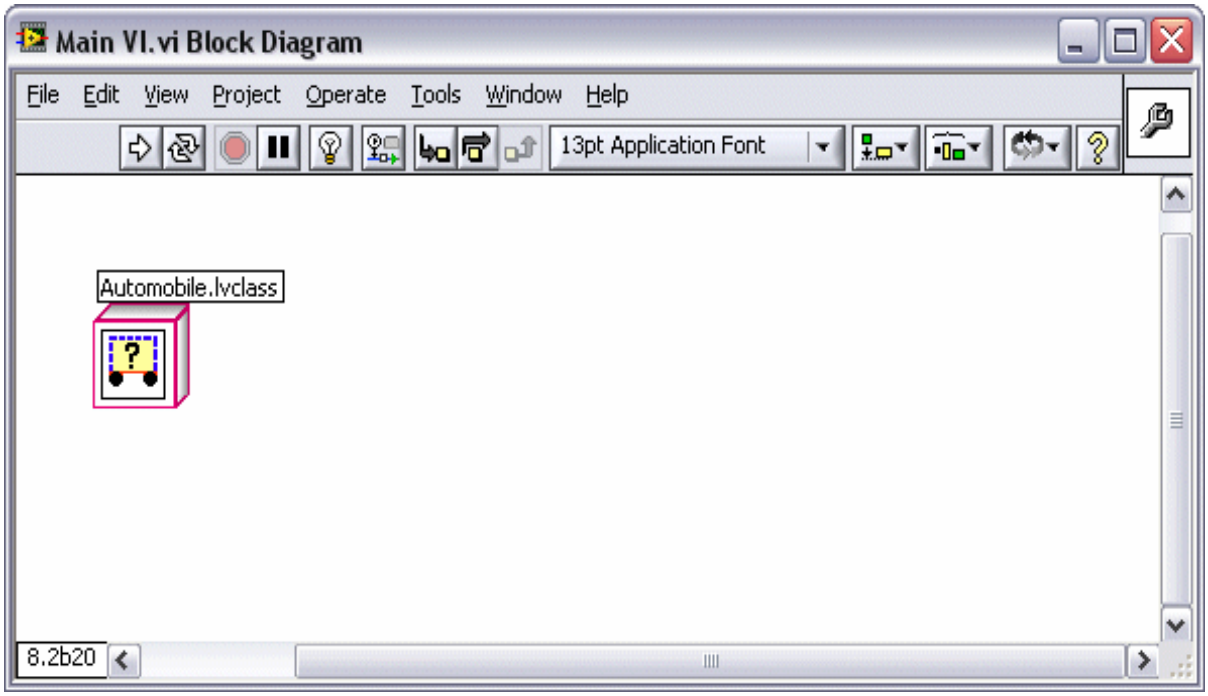
V objektově orientovaném programování, třídy reprezentují hlavní vlastnosti, které jednotlivě vytvořené objekty sdílejí. Můžete si třídy představit, obecně, jako jeden automobil. Automobily mají společné vlastnosti a třída tyto vlastnosti definuje. Objekt je specifický případ třídy. Objekt z třídy automobil může být konkrétní vůz, kterým jezdíte. Definice třídy určuje, jak se vaše auto, objekt chová.

Třída definuje data a metody spojené s objektem třídy. Dále používejme automobil jako příklad. Na světě existuje mnoho automobilů. Můžete obecně roztřídit všechny typy osobních nebo nákladních automobilů a autobusů, jako automobily. Automobily mají dveře a rychlosti. Informace o počtu dveří a počtu rychlostí jsou data o automobilu. Automobily můžou také zrychlovat a brzdit. Zrychlování a brzdění je chování neboli metody automobilu. Data a metody související s automobilem definují třídu Automobile. Vytvoříte třídu, která definuje data a metody daného objektu. Obr. 2.1 ukazuje LV třídu reprezentující třídu Automobile. LV ukládá data třídy Automobile v Automobile.ctl a metody třídy Automobile ve formátu *.vi s názvem Start Car.vi a Stop Car.vi.



Obrázek 2.1 Třída Automobile s metodami a privátními daty [2]

Objekt je konkrétní instance třídy. Automobil, kterým jezdíte, je zvláštní implementace třídy Automobile nebo objektu třídy Automobile. Blokové schéma Main VI, Obr. 2.2, zobrazuje objekt třídy Automobile. Objekty mají data a metody definované třídou. LV ukládá data třídy do ovládače třídy a vy vytvoříte VI, které implementují metody vámi vytvořené pro třídu LV.



Obrázek 2.2 Třída Automobile v blokovém diagramu [2]

2.1 VYTVOŘENÍ TŘÍDY

Vytvoříte uživatelsky-definované datové typy v LV vytvořením LV tříd. LV třídy definují data související s objektem, stejně jako metody, které definují operace, které můžete s daty provádět.

V LV, data třídy jsou soukromá, což znamená, že pouze VI, které jsou součásti této třídy mohou přistupovat k datům. Data třídy nadefinujete v ovladači privátních dat. Když vytvoříte a uložíte LV třídu, LV vytvoří soubor knihovny třídy (. Lvclass), která definuje nový datový typ. Soubor knihovny třídy uchovává ovladač soukromých dat a informací o nějaké součásti VI, kterou jste vytvořili, jako je např. seznam VI a různé vlastnosti VI. Knihovna třídy je podobná jako knihovna projektu (. Lvlib), nicméně knihovna třídy definuje nový datový typ.

Ovladač privátních dat je unikátní knihovna třídy, která definuje skupinu dat pro nové datové typy. LV neukládá ovladač privátních dat na disku. Namísto toho, LV jej uloží v rámci knihovny třídy. Ukládání ovladače soukromých dat uvnitř

knihovny třídy umožňuje, aby LV zajistilo, že byste nikdy neměli používat špatná privátní data definovaná třídou.

Tip: Můžete si vytvořit adresář na disku s názvem stejným jako LV třída pro uložení souboru knihovny třídy, součástí tříd VI a vlastních standardních sond. Ty mohou obsahovat soubory, které vlastní knihovna tříd v adresáři. Pokud máte hlavičkové soubory pro více než jednu knihovnu třídy ve stejném adresáři, mohly by nastat problémy v případě pokusu o zahrnutí VI se stejným názvem v různých knihovnách tříd. Jmenované problémy mohou nastat v procesu vývoje, když potlačíte dynamické VI.

2.2 ZAPOUZDŘENÍ

Každá LV třída se skládá z dat a metod. Data LV třídy jsou vždy privátní nebo skrytá před VI, které nejsou členy třídy. Chcete-li získat přístup k privátním datům, můžete vytvořit metody, a to v podobě VI ze třídy, k výkonu funkce na privátních datech ve třídě. Zapouzdření je sloučení dat a metod do třídy, kde jsou data přístupná pouze prostřednictvím VI třídy. Zapouzdření umožňuje vytvářet modulární bloky kódu, které můžete snadno aktualizovat nebo změnit, aniž by byly ovlivněny ostatní úseky kódu v rámci aplikace.

Ačkoli data ve třídě jsou vždy soukromá, můžete vystavit VI, který bude pro uživatele v různém stupni. Můžete mu nastavit přístup, rozsah a způsob na následující možnosti:

Veřejný prostor - Jakýkoliv VI může zavolat VI jako subVI.

Chráněná oblast působnosti - Pouze VI ve stejné třídě, nebo potomek třídy může zavolat VI. Chráněné VI zobrazí tmavě žlutý znak klíče v okně *Project Explorer*.

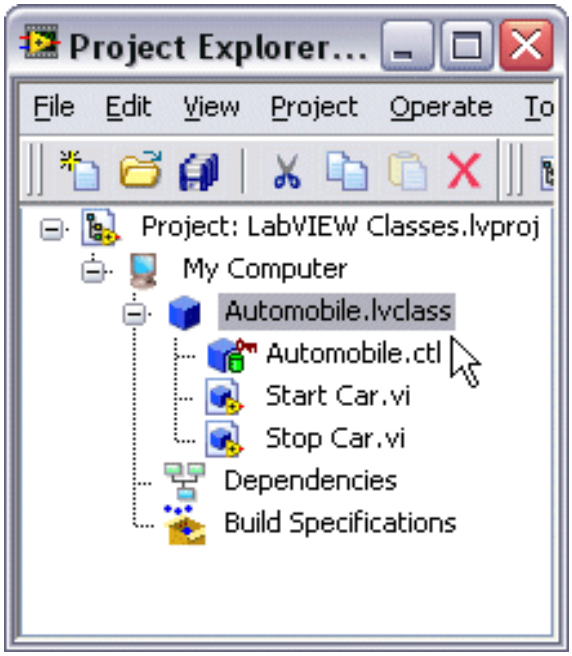
Soukromý rozsah -Pouze VI ve stejné třídě mohou volat VI. Soukromý VI zobrazí červený znak klíče v okně *Project Explorer*.

Vývojáři aplikací, které využívají LV třídy, nebo LV třídu uživatelů, mohou vytvořit VI mimo LV třídu a použít veřejného VI jako subVI na blokovém diagramu.

Veřejný VI umožní uživatelům LV třídy manipulovat s privátními daty ve třídě. Vývojáři, kteří vytvářejí LV třídy, mohou využít privátní a chráněné VI v blokovém diagramu VI k manipulaci s privátními daty ve třídě, která není vystavena uživateli LV třídy. Omezení vstupních bodů do třídy může usnadnit pro všechny vývojáře a programátory, ladění kódu, protože snížíte možnost pro zavedení chyb do dat.

2.3 DEFINOVÁNÍ OVLADAČE PRIVÁTNÍCH DAT

LV vytváří ovladač privátních dat ve třídě automaticky, když vytvoříte LV třídu. V okně *Project Explorer* uvedeného na Obr. 2.3 si všimněte, že ikona LV třídy je barevná kostka. Krychle představuje LV třídu. Ikona ovladače soukromých dat je barevná krychle se zeleným válcem. Válec představuje úložiště dat. Ikona ovladače soukromých dat také zobrazí znak červeného klíče označující, že je soukromý.

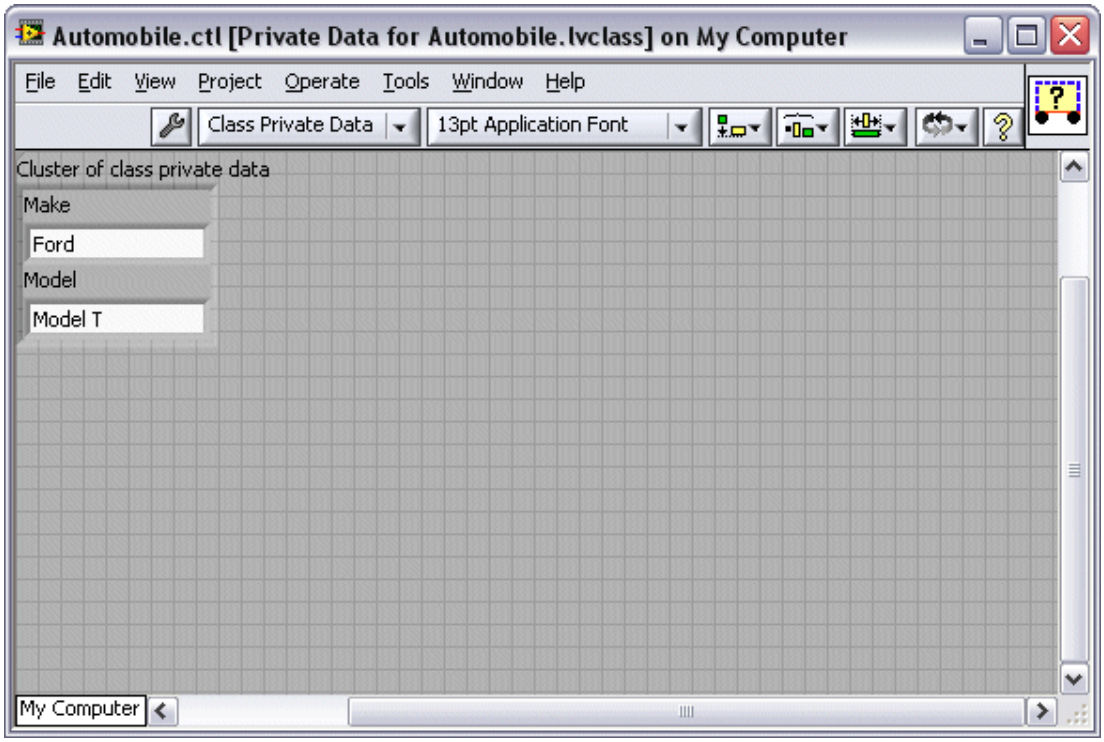


Obrázek 2.3 Okno Project Exploreru zobrazující třídu a metody [2]

Pomocí okna *Control Editor* upravujete ovladač privátních dat třídy. LV zobrazí okno *Control Editor*, když dvakrát kliknete na ovladač privátních dat třídy v okně *Project Explorer*. Můžete umístit ovládací prvky a indikátory do oblasti

Cluster of class private data (struktura privátních dat třídy) k definici typu privátních dat LV třídy. Výchozí hodnoty, které jste zadali ovládacím prvkům v *Cluster of class private data* jsou výchozí hodnoty pro tuto třídu.

V následujícím příkladu na Obr. 2.4 je datový typ automobilové třídy a obsahuje dva řetězce *Make* (značka) a *Model* (model).



Obrázek 2.4 Okno Control Editoru k editaci priv. dat třídy Automobile [2]

Poznámka: Můžete zanechat *Cluster of class private data* prázdný, pokud třída nepotřebuje žádná privátní data.

2.4 VYTVOŘENÍ VI

Vytvoříte VI nebo metody, které mají provádět operace na privátních datech ve třídě. VI jsou součástmi této LV třídy, ve které je vytvoříte a objeví se v okně *Project Explorer* pod ovladačem privátních dat ve třídě. Můžete definovat většinu metod pomocí jednoho VI v jedné třídě, ale některé metody si můžete definovat vytvořením vícenásobného VI po celé hierarchii třídy.

Vzhledem k tomu, že LV definuje data třídy, jako cluster (strukturu), můžete použít specifické funkce LV v blokovém diagramu VI pro přístup a manipulaci s daty. Použijte funkcí *Unbundle* a *Unbundle by Name* pro rozvázání clusteru privátních dat ve třídě na blokovém schématu VI. Pomocí funkcí *Bundle* a *Bundle by Name* znovu svažte cluster soukromých dat po vašem přístupu a manipulaci s ním. Vzhledem k tomu, že data třídy jsou privátní, svazující a rozvazující uzly přeruší provádění programu, pokud se pokusíte použít je s daty třídy v blokovém schématu nečlenského VI.

Můžete si vytvořit VI z šablon VI, která zahrnuje obsluhu chyb a tříd objektů z prázdného VI nebo z předchůdce VI. Pravým tlačítkem klikněte na třídu a vyberte z následujícího menu položek:

New » VI - Otevře prázdný VI.

New » VI from Dynamic Dispatch Template - LV doplní nový VI clusterem *error in* a *error out*, Case strukturou pro ošetření chyb, vstupy a výstupy LV třídy.

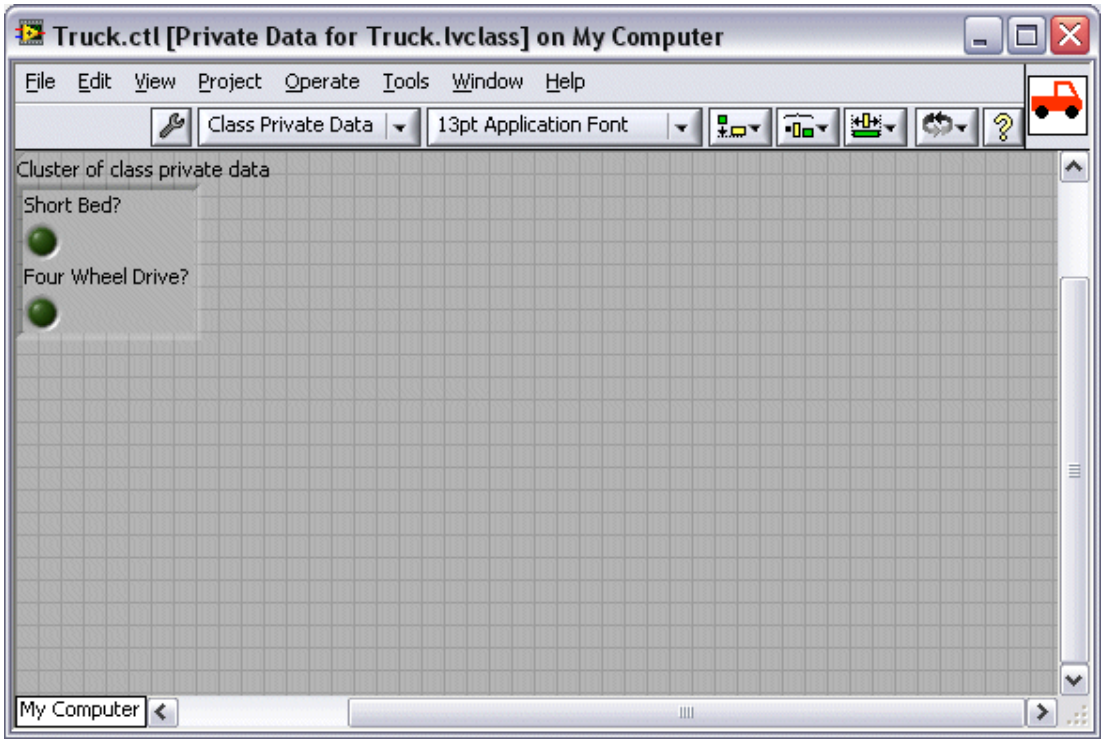
New » VI for Override - Umožní vám převýšit předka VI.

Poznámka: LV znepřístupní možnost *New » VI for Override*, pokud tam není platný VI k převýšení.

2.5 DĚDĚNÍ

Dědění vám umožní použít již existující třídy jako výchozí bod pro novou třídu. Pokud si vytvoříte novou třídu LV a nastavíte jí dědit data a VI z jiné třídy, nová třída může využívat veřejné a chráněné VI ze třídy, ze které je zdědí. Je také možné přidat svá vlastní data a VI, aby se zvýšila jeho funkčnost. V následujícím příkladu privátní data z třídy Automobile obsahují *Number of Gears, Number of Doors, Make* a *Model*. Pokud si vytvoříte novou třídu a pojmenujete ji Truck, můžete nastavit Truck k dědění dat z třídy Automobile, stejně jako přidat Booleovská data *Short Bed?* (krátká postel?) a *Four Wheel Drive?* (pohon na čtyři kola). Privátní data

třídy Truck nyní obsahují *Number of Gears Number of Doors, Make, Model, Short Bed?* a *Four Wheel Drive?*, jak je zobrazeno na Obr. 2.5.



Obrázek 2.5 Privátní data potomkovské třídy Truck [2]

Když svážete nebo rozvážete LV třídu, uzly zobrazí vývody pouze pro privátní data aktuální třídy, ne pro všechna data třídy zděděná od předka třídy. Data předka jsou privátní a vy je modifikujete pomocí funkcí předka třídy poskytovaných prostřednictvím VI. VI potomků tříd mohou volat některý z veřejných VI, stejně jako jakékoli VI v LV. Ale VI potomka třídy také může zavolat chráněné VI předka třídy. Když určíte jeden VI předka jako chráněný, VI jakéhokoli potomka třídy může volat metody, ale žádná VI mimo dědickou hierarchii, tak nemohou učinit. Pokud chcete mít přístup na *Numer of Gears* třídy Automobile ve třídě Truck, můžete vytvořit veřejný nebo chráněný VI ve třídě Automobile volající *Get Gears.vi*. Můžete otevřít třídu Automobile na blokovém schématu *Get Gears.vi*, odhalující *Numer of Gears*. Můžete pak přiřadit *Number of Gears* na výstupní svorku konektoru panelu, který

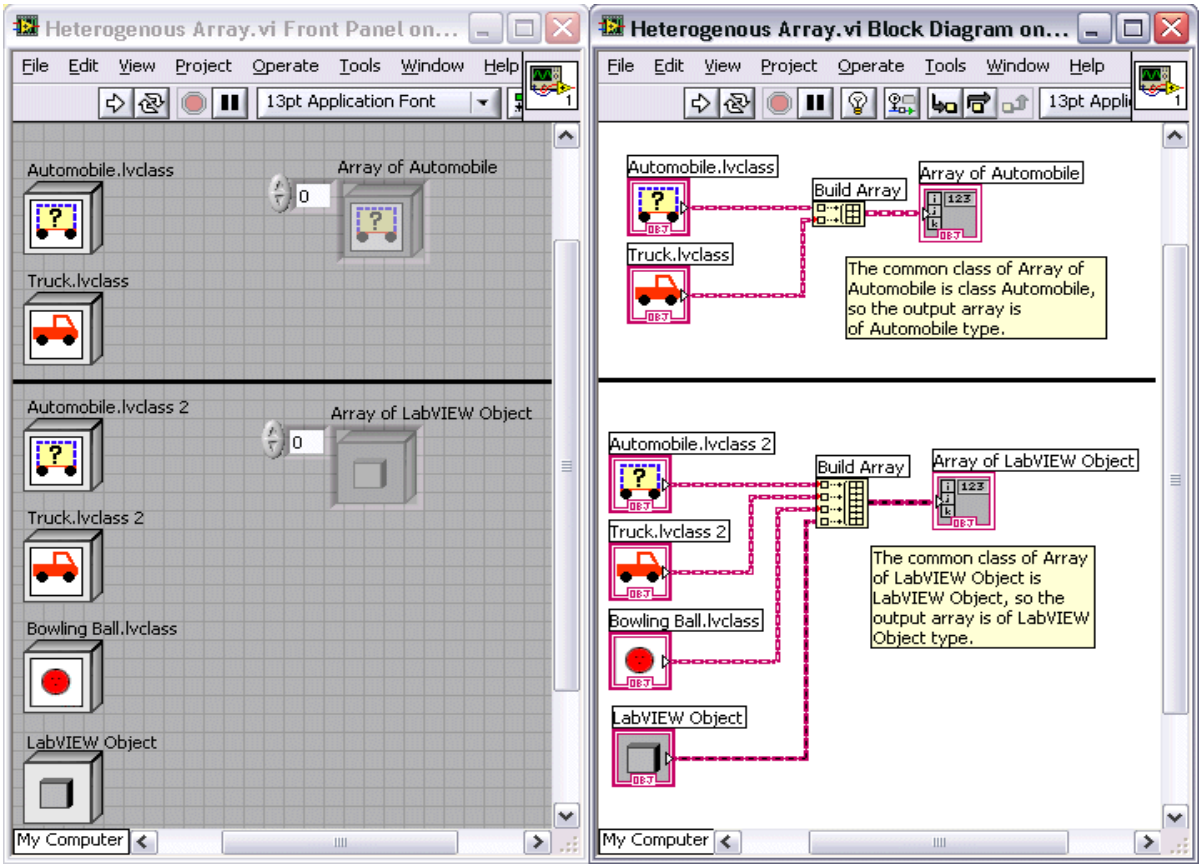
vám umožní přístup k určitým privátním datům ve třídě Automobile v potomkovi třídy, jako je třída Truck.

Poznámka: LV třídy nemohou volat privátní VI jiné třídy LV dokonce ani z mateřské třídy. Můžete použít privátní VI pouze v blokovém schématu z jiných VI uvnitř té samé třídy.

3. LABVIEW OBJEKT

Výraz LV objekt je název konkrétní třídy. LV Objekt je konečným předek ze stromu dědičnosti v objektově orientovaném programování v LV. Ve výchozím nastavení všechny LV třídy dědí z LV Objektu. Můžete použít LV Objektu k vytvoření VI, které může provádět obecné operace na více paralelních LV třídách. Například, pokud si vytvoříte pole LV tříd, data z pole jsou různorodá, protože mohou obsahovat prvky ze třídy typu pole nebo jakéhokoli potomka třídy. Pokud je pole typu Objekt LV, může obsahovat Automobile, Truck a Bowling Ball. Třída Bowling Ball nemá dědit ze třídy Automobile nebo Truck a tak LV vytvoří pole nejvíce společných předků základní třídy, v tomto případě LV Objekt.

Následující Obr. 3.1 zobrazuje *Array of Automobile* (pole automobilů), což je pole, které obsahuje třídu Automobile a třídu Truck. Protože Truck dědí z Automobile je společný předek základní třídy tohoto pole typ Automobile. Obrázek také zobrazuje *Array of LabVIEW Object* (pole LV objektů), které obsahuje třídy LV Objekt, Automobile, Truck a Bowling Ball. Bowling Ball nedědí z Automobile ani Truck, ale všechny tři nakonec dědí z LV Objektu, proto tedy *Array of LabVIEW Object* je typu LV Objekt.



Obrázek 3.1 Pole tříd a pole objektů [2]

3.1 NASTAVENÍ DĚDĚNÍ

Všechny LV třídy standardně dědí z LV objektů. Pokud chcete změnit třídu, ze které třída dědí musíte změnit dědění po vytvoření třídy. Můžete nakonfigurovat dědění a další možnosti této třídy v dialogovém okně *Class Properties*. Můžete si prohlédnout hierarchii LV tříd v okně *LabVIEW Class Hierarchy*. Hierarchie dědění tříd mohou zahrnovat:

Rodičovská třída - třída LV, ze které ostatní LV třídy dědí data a veřejné a chráněné VI.

Dítě třídy - LV třída, která převezme data a veřejné a chráněné VI z mateřské třídy.

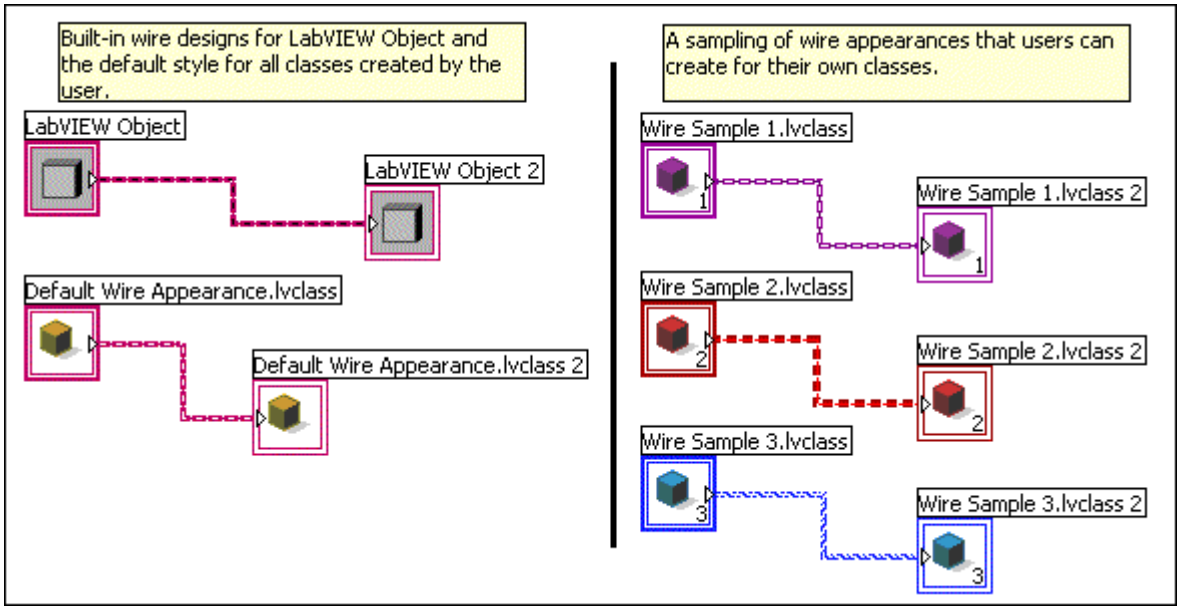
Sourozenecká třída - LV třída, která sdílí stejné mateřské třídy jako jiná LV třída.

Třída předka - LV třída, která je rodič, prarodič, pra-prarodič a tak dále. LV Objekt je konečným předkem všech LV tříd.

Potomek třídy - LV třída, která je pro dítě, vnouče, pravnouče, a tak dále.

3.2 VZHLED SPOJŮ

Třídy definují nové datové typy. Spoje těchto typů tříd se zobrazí na blokovém schématu jako spoj výchozí LV třídy nebo zděděný vzhled spoje mateřské třídy. Můžete změnit vzhled spoje LV třídy v dialogovém okně *Class Properties*. K vytvoření snadno čitelného blokového diagramu, můžete změnit vzhled spojů různých LV tříd, když je to vhodné. Nadměrný počte barev spojů a vzorů může znamenat, že blokové schéma bude hůře čitelné. Následující obrázek,Obr. 3.2 , ukazuje vestavěné vzhledy spojů LV vlevo a některé možnosti vlastních vzhledů spojů na pravé straně.



Obrázek 3.2 Ukázky spojů výchozích a vlastních [2]

4. DYNAMICKY A STATICKY ODESÍLAJÍCÍ VI

Metoda je operace definovaná LV třídou. Můžete definovat několik metod s jediným VI. Ty se nazývají *statické metody*, protože LV volá pokaždé stejné VI. Můžete také definovat metody pro více VI se stejným názvem v celé hierarchii třídy. Ty se nazývají *dynamickými metodami*, protože, který z VI LV volá, není známo až do času zpracování. Dynamické metody jsou podobné jako polymorfní VI. Kde polymorfní VI určí, které VI volá, to závisí na typu dat, které jste připojili, dynamické metody čekají až do zpracování s určením, který VI v hierarchii volá a to závisí na datech, které dorazí na vstupní svorky.

Vy určíte VI buď jako statický nebo dynamický na konektorech bloků VI. Pokud konektor bloku obsahuje dynamicky odesílající vstupní svorky, VI je součástí dynamické metody. Pokud tam nejsou dynamicky odesílající vstupní svorky, VI definuje statické metody.

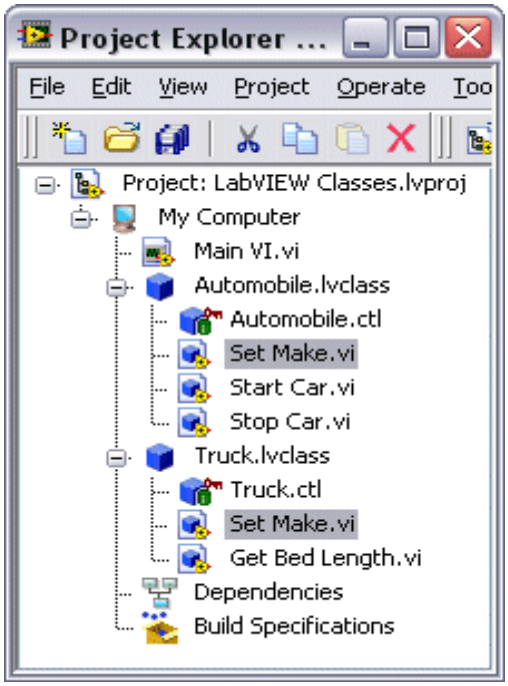
Když jedna LV třída dědí z jiné třídy, potomek třídy zdědí všechny veřejné a chráněné metody definované v mateřské třídě. Pojmenováním VI ve třídě potomka přesně stejným jménem jako má VI v mateřské třídě, můžete definovat potomkovskou implementaci metody.

Vzhledem k tomu, že LV definuje statické metody, pomocí jediného VI, nepojmenovávejte VI potomka třídy stejným jménem jako statický VI ve třídě předka. Například, pokud rodič třídy Automobile obsahuje statické VI, Open Door.vi, pak dítě třídy Truck nemůže mít VI s názvem Open Door.vi, protože metoda je již definována v třídě Truck, jelikož Truck dědí VI z Automobile. Pokud umístíte statické metody na blokové schéma jako subVI, chovají se ve všech směrech jako běžné volání subVI.

Můžete definovat vícenásobné dynamické VI pro metodu, po jednom na každé úrovni v hierarchii dědičnosti. V následujícím obrázku Obr. 4.1, třída Automobile a třída Truck jsou obě definované dynamickou metodou Set Make.vi. Pokud umístíte dynamický VI na blokové schéma jako subVI, uzel na blokovém schématu se chová stejně jako běžné subVI volání, když je LV v editovacím modu. Pokud jakkoli spustíte VI, data, která jdou do dynamicky odesílající vstupní svorky

určí, který z VI z hierarchie třídy LV zavolá. Protože spoj LV třídy může vést data svého vlastního typu nebo data nějakého potomkovského typu, uzel vykonávající kterýkoliv VI definujete pro data třídy.

Poznámka: LLB nemohou obsahovat soubory se stejným názvem. Proto, pokud máte dynamické VI, které sdílejí jména v hierarchii tříd, nemůžete dát tyto třídy do stejného LLB.



Obrázek 4.1 Ukázka dyn. metody Set Make [2]

Poznámka: Pokud definujete vícenásobné dynamické VI pro stejnou metodu na různých úrovních hierarchie, všechny VI metody, musí mít stejné *reentrant settings* (nastavení vícenásobného přístupu), *preferred execution settings* (nastavení preferovaného spouštění), *priority settings* (nastavení priority), *connector pane terminal* (svorky konektoru bloku) a *access scope* (rozsahu přístup).

Pokud vytvoříte převyšující VI výběrem **New » VI**, můžete vytvořit dynamicky odesílající VI, protože VI, které převyšuje předka VI má stejné jméno jako předek, a také obsahuje dynamicky odesílající svorky. LV umístí *Call Parent*

Method Node (uzel volání mateřské metody) na blokové schéma s příslušnými dynamickými vstupními a výstupními svorkami třídy a jinými svorkami VI potřebnými, aby odpovídaly předkovi VI. LV zakáže *Override VI* (převyšující VI) volbu neexistuje-li předek VI k převýšení.

4.1 DYNAMICKÉ VÝSTUPY

Můžete označit výstupní svorky LV třídy jako dynamické kliknutím pravého tlačítka na svorku bloku a výběrem *Dynamic Dispatch Output (Recommended)* (dynamické odesílání výstupu (doporučeno)). Když voláte VI přes dynamickou výstupní svorku jako subVI dynamický, výstup se změní na stejný datový typ jako je připojen k dynamické vstupní svorce. Například pokud připojíte třídu Automobile na dynamicky odesílající vstupní svorku, výstup VI je tentýž jako na vstupu; v tomto případě bude výstup třída Automobile. Můžete měnit data mezi dynamickou vstupní svorkou a dynamickou výstupní svorkou. Nicméně, aby byl zajištěn bezpečný průběh programu v třídě LV, data z dynamické vstupní svorky musí téct do všech dynamických výstupních svorek. Také k zajištění, že LabVIEW čte z dynamické vstupní svorky přesně jednou, a zapisuje na dynamickou výstupní svorku přesně jednou, nemůžete umístit dynamické svorky předního panelu uvnitř struktury.

Poznámka: Pokud ladíte dynamický VI s dynamickými vstupy a výstupy, můžete zkoumat spoj, který vede z chybového dynamického vstupu do chybového dynamického výstupu. Barva pozadí spoje je šedá místo obvyklé bílé pro každý spoj, který začíná v dynamickém vstupu a neprochází přes žádnou funkci, která může změnit datový typ za běhu programu. Barvu pozadí spoje zčervená, pokud spoj prochází funkcí, která může změnit datový typ. Aby dynamický výstup správně pracoval, nemůžete měnit datový typ LabVIEW třídy.

Pokud chcete dokončit operaci na blokovém schématu VI, poznáte, že výsledkem v LabVIEW třídě je výstupní datový typ, který je jiný než vstupní, ujistěte se, že dynamická výstupní svorka konektoru bloku je nastavena na

Recommended (doporučené) namísto *Dynamic Dispatch Output (Recommended)* (dynamicky odesílající výstup (doporučeno)). Například, pokud je vstupem třídy Automobile a víte, že chcete na výstupu třídu Truck, měli byste změnit LabVIEW třídě výchozí svorku na konektoru bloku. Případně můžete vytvořit VI pomocí prázdného VI, který vám umožní nastavit ručně svorky konektoru bloku.

Poznámka: Používáte-li strukturu Case nebo Event v některém z členských VI s dynamickým vstupy a výstupy, musíte ručně propojit tunel ve všech případech struktury. Pokud použijete *Use Default if Unwired* (použít výchozí pokud nezapojeno) na výstupu tunelu, LabVIEW přeruší VI.

5. JAKÉ JSOU ROZDÍLY MEZI OOP V C ++ A LABVIEW?

Zatímco LV objektově orientované programování je podobné ostatním objektově orientovaným jazykům koncepčně, LV je grafické programovací prostředí řízené tokem dat a proto existují některé rozdíly ve způsobu jakým LV ovládá a komunikuje s daty třídy a jak můžete ladit kód třídy.

5.1 ZÁKLADNÍ ROZDÍLY PRINCIPU FUNKCE

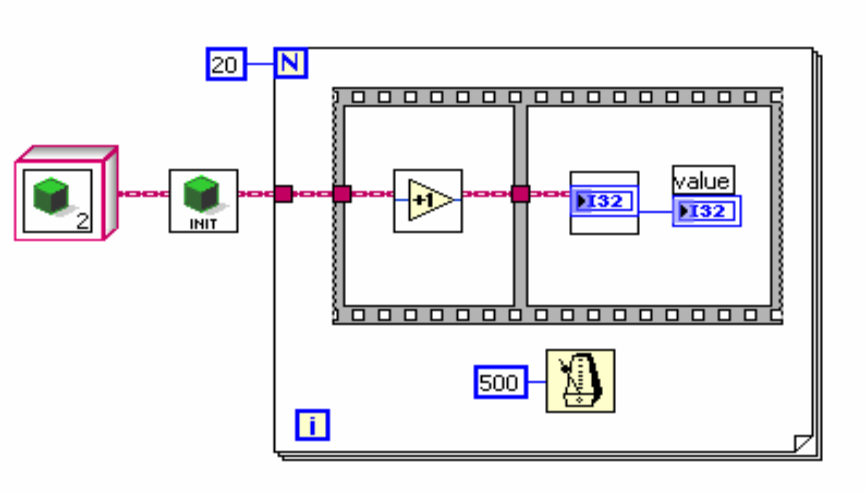
- LV je konečným předek třídy pro všechny objekty. C + + není.
- C + + má konstruktory. LV žádné nepotřebuje.
- C + + má destruktory. LV žádné nepotřebuje.
- C + + má obě, referenční i hodnotovou, syntaxi pro předávání objektů jako parametrů. LV má pouze hodnotovou syntaxi, s odkazy vytvořenými prostřednictvím dalších funkcí.
- LV má automatický převod dat, takže uživatel může získat stará data, i když ve třídě byla upravována.
- C + + nemá. Požaduje se, aby uživatel sledoval změny v převodech a napsal převodní kód pro data.
- C + + má šablony. LV nemá.
- C + + má čisté virtuální funkce. LV 8.2 a 8.5 nemá.
- C + + má několik typů dědění. LV jen jediný, veřejné dědění.

Nyní se podívejme na názornou ukázku některých odlišností v jednoduché úloze vypracované v C++ a LabVIEW. Za úkol máme hodnotu třídy obsaženou v jednoprvkovém poli FOR-cyklem o 20 krocích postupně inkrementovat vždy o 1 v každém kroku. Nejprve si zopakujme jak by to vypadalo v C++.

```
class myClass
      {
       private:
         int myMember;

       public:
         myClass()
         {
          // konstruktor - inicializace
          myMember = 0;
         }

         int increment ()
         {
          // inkrementace o 1
          myMember = myMember +1;
          return 0;
         }
```

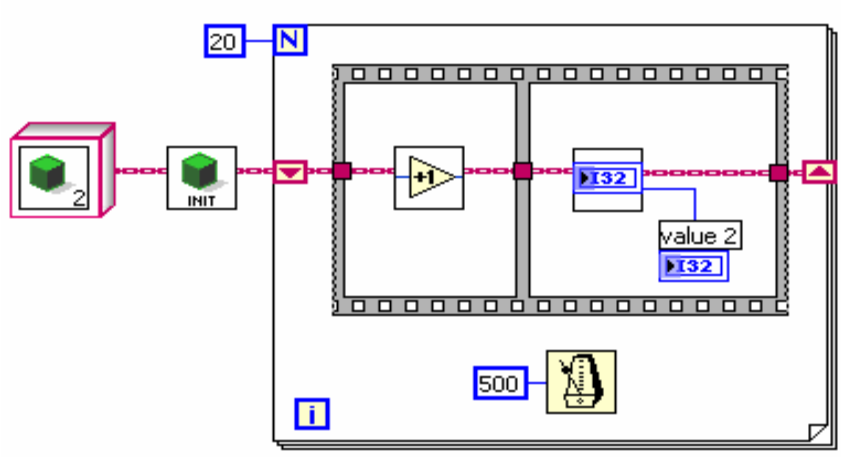
```
int output()
    {
    // výpis prvku
    printf("%d", myClass.myMember);
    return 0;
    }
}

int Main()
    {
    myClass myObject = new
myClass(0);

    for (int i = 0; i < 20; i++)
    {
    myObject.increment();
    myObject.output();
    }
    }
```

A nyní přesně to samé implementované do LV, tak jak ukazuje následující obrázek Obr. 5.1. Jenomže zjistíme, že mechanismus, který fungoval v C++ nefunguje v LV. Těm, kteří přecházejí z C++ na LV, se to jeví nepochopitelné, ale je to tak. Program proběhne sice 20krát, ale vždy se vezme prvek inicializovaný nulou a zvýší se o 1, takže výsledkem je vždy hodnota 1 a ne požadovaných 20. V LV mechanismus funguje tak, že se do třídy zapíše až finální hodnota po skončení celého cyklu, nikoli po každém kroku.



Obrázek 5.1 Nefunkční implementace do LabVIEW [1]

Abychom daný problém odstranili, musíme ve FOR-cyklu využít tzv. *Shift Register* (posuvný registr), který má tu funkci, že nám přenese výslednou hodnotu předchozího kroku do následujícího kroku smyčky a my s ní pak můžeme dále pracovat. Tím docílíme toho, že vše bude fungovat tak jak má, v prvním kroku se inicializovaná nulová hodnota zvýší na 1, ve druhém kroku na 2 atd. až do hodnoty 20.



Obrázek 5.2 Funkční implementace do LabVIEW [1]

6. OOP VERSUS KLASICKÝ PŘÍSTUP

Objektově orientované programování prokazuje svou převahu nad procedurálním programováním jako strukturovaný výběr z ostatních programovacích jazyků. Podporuje čistší rozhraní mezi jednotlivými částmi kódu, je snadnější k ladění, a proto tomu programátorské týmy přikládají takovou váhu.

Myšlenka objektově-orientovaného programování je, že počítačový program může být vnímán jako kolekce zahrnující jednotlivé celky nebo objekty, které spolupracují navzájem, na rozdíl od tradičního pohledu, ve kterém může být na program nahlíženo jako na kolekci funkcí nebo jednoduše jako seznam instrukcí počítače. Každý objekt je schopen přijímat zprávy, zpracovávat data a posílat zprávy do jiných objektů. Každý objekt může být vnímán jako nezávislý malý stroj.

O objektově orientovaném programování se prohlašuje, že prosazuje větší flexibilitu a udržovatelnost při programování a je velmi rozšířené v celé škále softwarového inženýrství. Kromě toho, zastánci OOP tvrdí, že je jednodušší pro nováčky v programování se jej naučit, než v předchozích programovacích přístupech a že OOP je často jednodušší přístup k rozvoji, kódování a pochopení složitých situací a postupů, něž je tomu u jiných metod programování.

Nicméně, v LabVIEW to není všelék, mnoho programátorů jistě potvrdí, že nastanou situace, kdy OOP opravdu nenabízí nějaké výhody proti starším metodám.

7. SHRNUTÍ

Co říci závěrem? Objektově orientované programování v LabVIEW si jistě najde své uplatnění i příznivce. Kladem grafického programovacího rozhraní oproti textovým programovacím jazykům je jistě větší přehlednost a názornost, protože LabVIEW je řízeno tokem dat, tak vidíme, kterými bloky VI jsou data postupně vedena a tak lépe pochopit strukturu programu. Naopak k nepřehlednosti přispívá to, že tvoříme-li obsáhlejší program a máme otevřeno více oken VI najednou, je obtížnější orientace.

Problém obklopující OOP v LV je jeho současná realizace OO principů. National Instruments rozhodlo jaké komponenty do OOP implementovat. Věřili, že komponenty jimi implementované jsou nejdůležitější, ale je to diskutabilní. Musí se objektivně říci, že LV OOP není nejlepší implementací OO principů, která doposud existuje. Neznamená to ovšem, že je LV OOP "bezcenné".

Jistě ale ještě nějakou dobu potrvá než se vychytají všechny mouchy a implementace OO principů se ustálí na nějakém, řekněme standardu, aby byly mechanismy v C++ a LabVIEW rovnocennější a snáze pochopitelné pro zastánce obou stran problematiky.

8. ZDROJE IINFORMACÍ

[1] National Instruments, [online], [cit. 2008-12-8], <http://www.ni.com>

[2] LabVIEW [počítačový soubor nápovědy programu]. Ver. 8.5. Austin (Texas, USA): National Instruments, 2007. Dostupné z URL <http://digital.ni.com/manuals.nsf/websearch/F8C1BE5BC6D47DE28625732B0053510C>

9. SEZNAM ZKRATEK

Zkratka	Popis
LV	LabVIEW
VI	Virtuální přístroj
subVI	Virtuální přístroj uložený v knihovně
OO	Objektově orientované
OOP	Objektově orientované programování
NI	National Instruments
LLB	LabVIEW VI library

10. SEZNAM PŘÍLOH

1. Úloha do cvičení - MP3 players
2. Úloha do cvičení – Signal
3. CD obsahující zdrojový tvar práce a vypracované úlohy podle návodů

11. PŘÍLOHY

11.1 ÚLOHA DO CIVČENÍ – MP3 PLAYERS

11.1.1 Zadání

Navrhněte program, který bude pracovat s polem objektů typu MP3 přehrávač a jeho parametry (název, kapacita paměti, typ napájení, výdrž baterií, cena, FM tuner, slot na přídavné karty a diktafon) umožňující vstup i výstup to textového souboru.

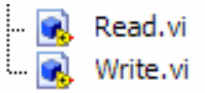
11.1.2 Postup

1) Spustíme program LabVIEW a zvolte možnost Empty Project pro zobrazení okna Project Explorer. V tomto okně v menu File>>New najdeme a označíme volbu Class, potvrdíme OK a pojmenujeme třídu např. MP3player.

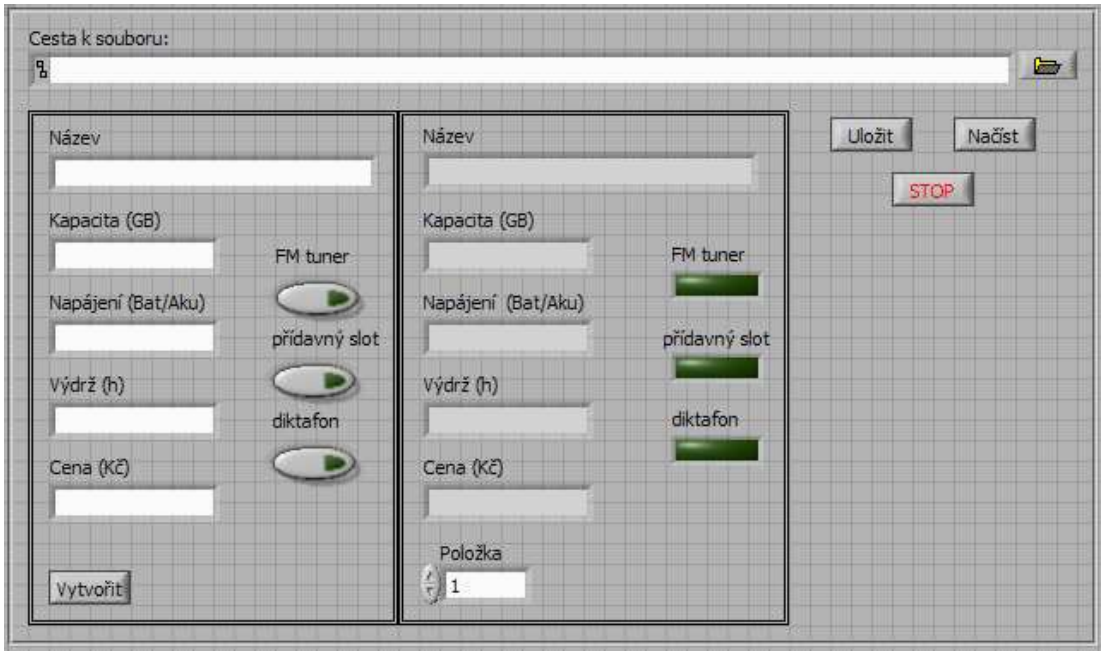
2) V Project Exploreru rozvineme položku MP3player.lvclass a poklepáním na MP3player.ctl se zobrazí okno s prázdným clusterem (strukturou) privátních dat třídy. Do zobrazeného clusteru vložíme 5 komponent String Indicator ze záložky Classic nebo Modern>>String a 3 komponenty LED sloužící k uchování log informací.

3) Aby bylo možné vykonat další krok, je třeba celý projekt uložit.

4) Z kontextové nabídky položky MP3player.lvclass zvolíme New>>VI from Static Dispatch Template. Použitím bloku Unbundle by Name a Indicatoru na všechny parametry obsažené ve třídě si vytvoříme metodu provádějící čtení všech těchto parametrů. Stejným způsobem se provede i zápis parametrů pouze místo Unbundle by Name se použije Bundle by Name a namísto Indicatoru zvolíme Control.

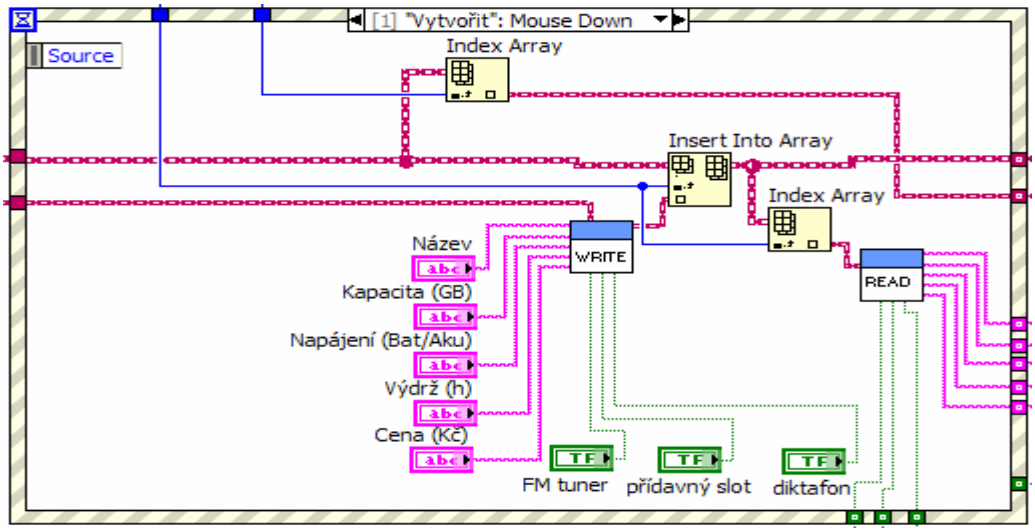


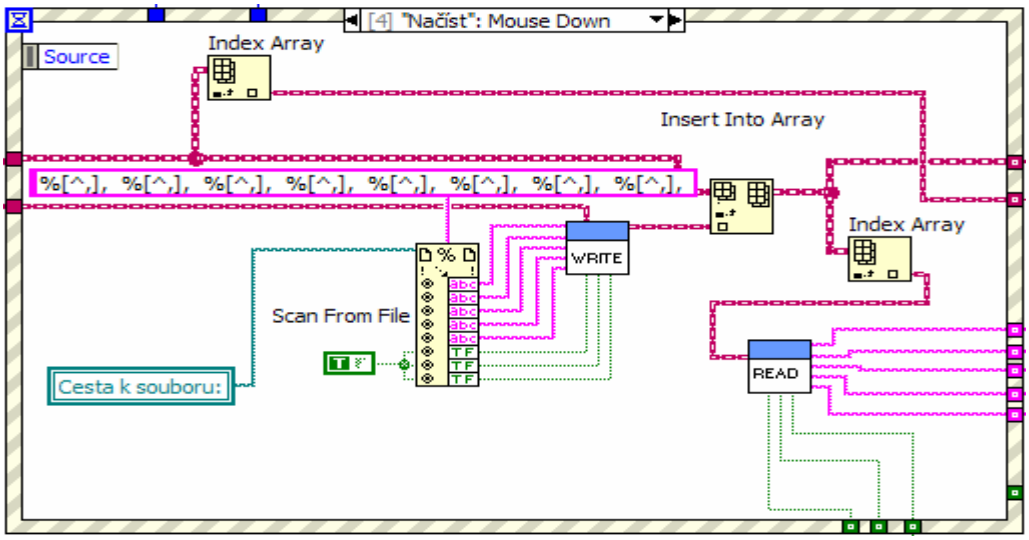
5) Uživatelské rozhraní může vypadat např. takto:

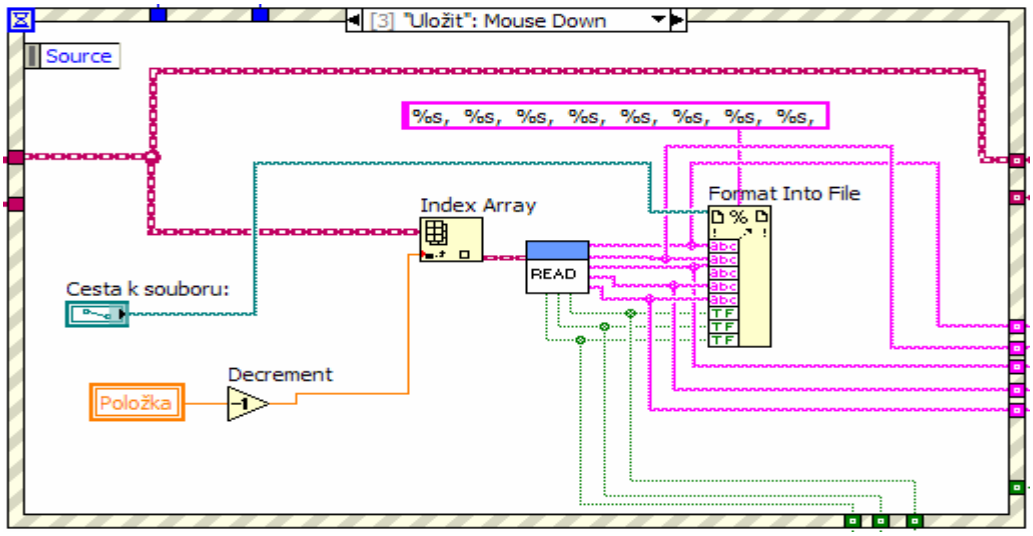


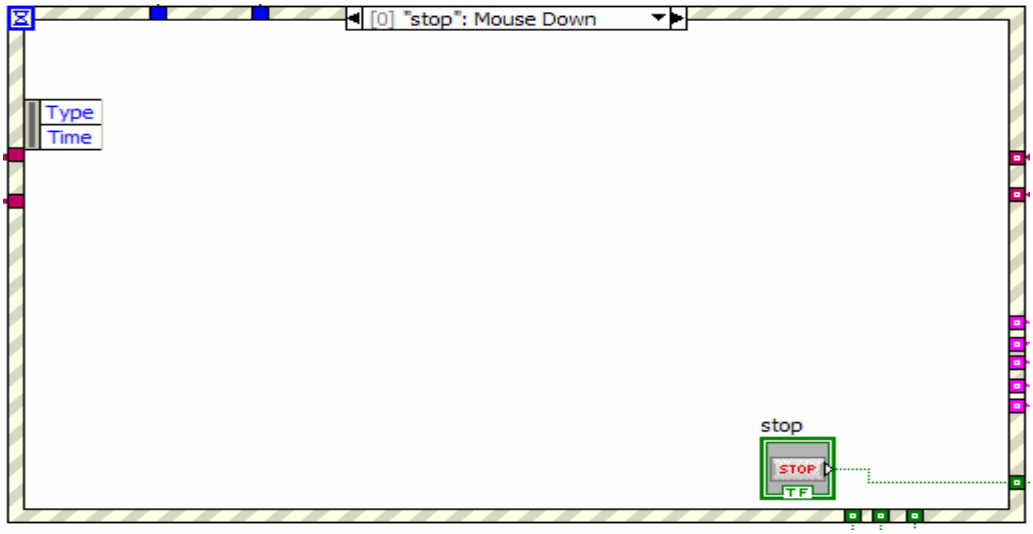
Komponenty v levém rámečku slouží k zápisu objektu do pole objektů a v pravém rámečku k procházení databáze. Do horního řádku se zapisuje cesta k souboru určeného k zápisu nebo načtení nějakého objektu.

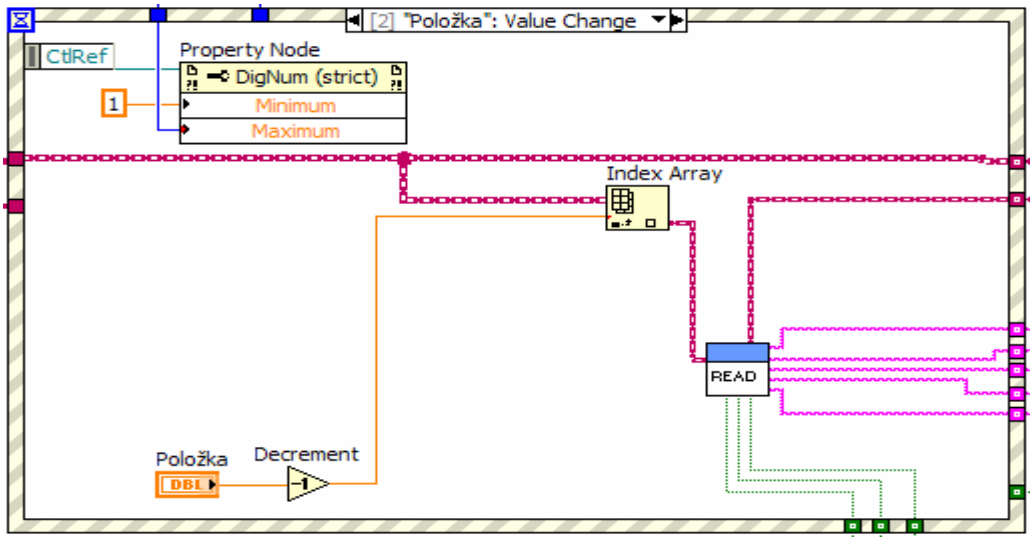
6) Vytvoříme hlavní program (Main.vi), celý program poběží v cyklu WHILE a jednotlivé události jsou ošetřeny EVENT strukturou. Těchto událostí je celkem 5 a jsou to: stisknutí tlačítek Vytvořit, Načíst, Uložit a STOP a ještě procházení jednotlivých položek. Blokový diagram událostí vidíme na obrázcích:



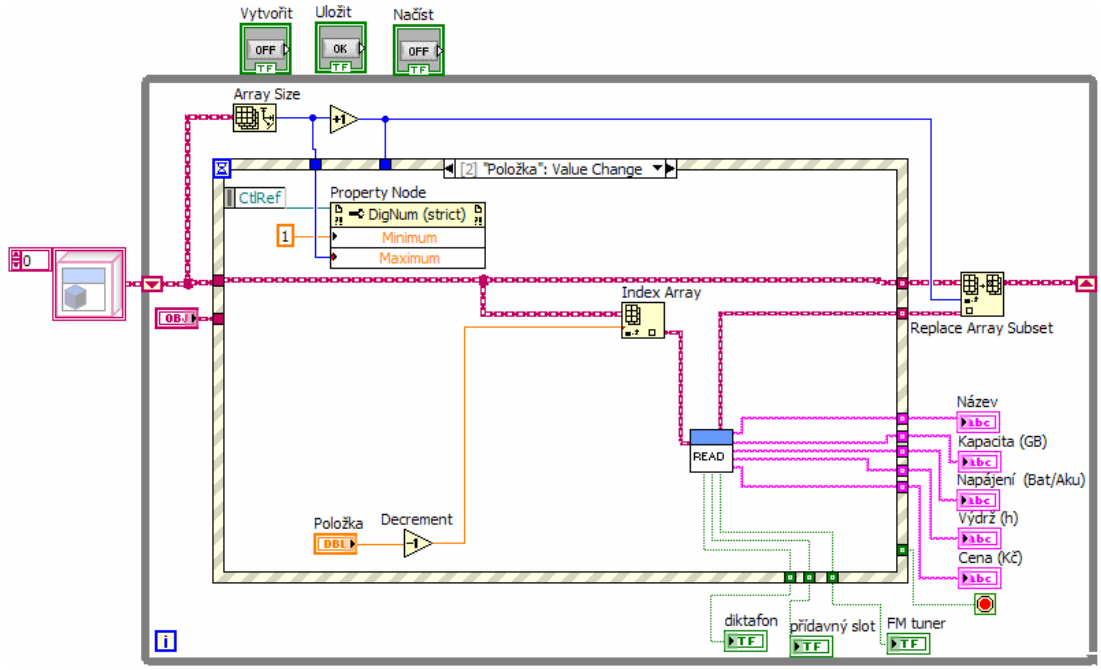

[4] "Načíst": Mouse Down
Source
Index Array
Insert Into Array
%[^,], %[^,], %[^,], %[^,], %[^,], %[^,], %[^,], %[^,],
WRITE
Index Array
Scan From File
Cesta k souboru:
READ


[3] "Uložit": Mouse Down
Source
%s, %s, %s, %s, %s, %s, %s, %s,
Index Array
Format Into File
READ
Cesta k souboru:
Decrement
Položka


[0] "stop": Mouse Down
Type
Time
stop
STOP



Celý blokový diagram Main potom vypadá následovně:



Z obrázků lze také vyčíst, že si vystačíme s metodami Read a Write a komponentami Index Array, Insert into Array, Replace Array Subret, Increment, Decrement, Property Node, Path Control, Format Into File, Scan From File a Local Variable.

11.1.3 Zhodnocení

Tuto úlohu bych zařadil do střední kategorie dle obtížnosti. Při jejím vytváření se procvičí práce s objekty resp. s poli objektů a práce se soubory tzn. zápis a načítání privátních dat objektů do formátu TXT.

11.2 ÚLOHA DO CVIČENÍ - SIGNAL

11.2.1 Zadání

Navrhněte třídu Signal, která bude realizovat knihovnu pro práci se signály. Tato třída bude obsahovat pole vzorků a bude umožňovat základní operace, jako je součet, rozdíl a součin signálů. Dále ještě výpočet střední a efektivní hodnoty výsledku po provedení operace. Pro demonstraci postačí signály sinusového průběhu a obdélníkového impulsu.

11.2.2 Postup

1) Spusťte program LabVIEW a zvolte možnost Empty Project pro zobrazení okna Project Explorer. V tomto okně najdeme v menu File >> New, najdeme a označíme volbu Class a potvrdíme tlačítkem OK a pojmenujeme třídu Signal.

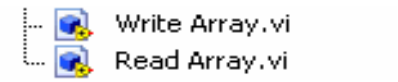
2) V Project Exploreru rozvineme položku Signal.lvclass a poklepáním na Signal.ctl se zobrazí okno s prázdným clusterem (strukturou) privátních dat třídy. Do zobrazeného clusteru vložíme komponentu Array z nabídky Control záložka Modern >> Array, Matrix & Cluster. Nyní je třeba určit co bude pole obsahovat. Do pole vložíme Numeric Control komponentu (stačí jedna, LabVIEW automaticky pole přizpůsobí podle toho, kolik vzorků zapíšeme do pole) ze stejné nabídky záložka Modern >> Numeric.

3) Aby bylo možné vykonat další krok, je třeba celý projekt uložit. A nyní můžeme přistoupit k vytváření metod neboli operací s poli.

4) Z kontextové nabídky položky Signal.lvclass zvolíme New >> VI for Data Member Acces.

Zde máme možnost zvolit cíl přístupu (privátní data třídy), dále Acces jestli chceme vytvořit VI pro zápis či čtení a ještě zda vytvořený VI bude mít dynamický nebo statický přístup.

My zvolíme následující nastavení, Array pro přístup k poli jako k celku, Acces požadujeme Read and Write a v našem případě bude stačit statický přístup, volby potvrdíme tlačítkem Create. Všimněme si, že v Project Exploreru se objevily nově vytvořené VI, které dvojklikem můžeme otevřít a prohlédnout si jak pracují, případně upravit jejich funkci.



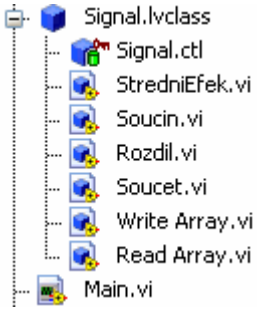
5) Nyní přistoupíme k vytvoření samotných operací s poli. Ze stejné kontextové nabídky jako v předchozím případě vybereme VI from Static Dispatch Template a pojmenujeme ho Soucet.

Použitím subVI Read Array a komponenty pro sčítání (Add) a subVI Write Array vytvoříme VI pro operaci součtu polí tříd Signal a výsledek uložíme do další prázdné třídy. Zde je ale potřeba upravit konektor svorek vytvořeného VI. To provedeme tak, že klikneme pravým tlačítkem na obrázek ikony VI vpravo nahoře, volbou Show Connector a výběrem neobsazeného místa na panelu svorek a označením ikony na VI zastupující třídu pro výsledek resp. pro třídu druhého signálu vstupujícího do operace.

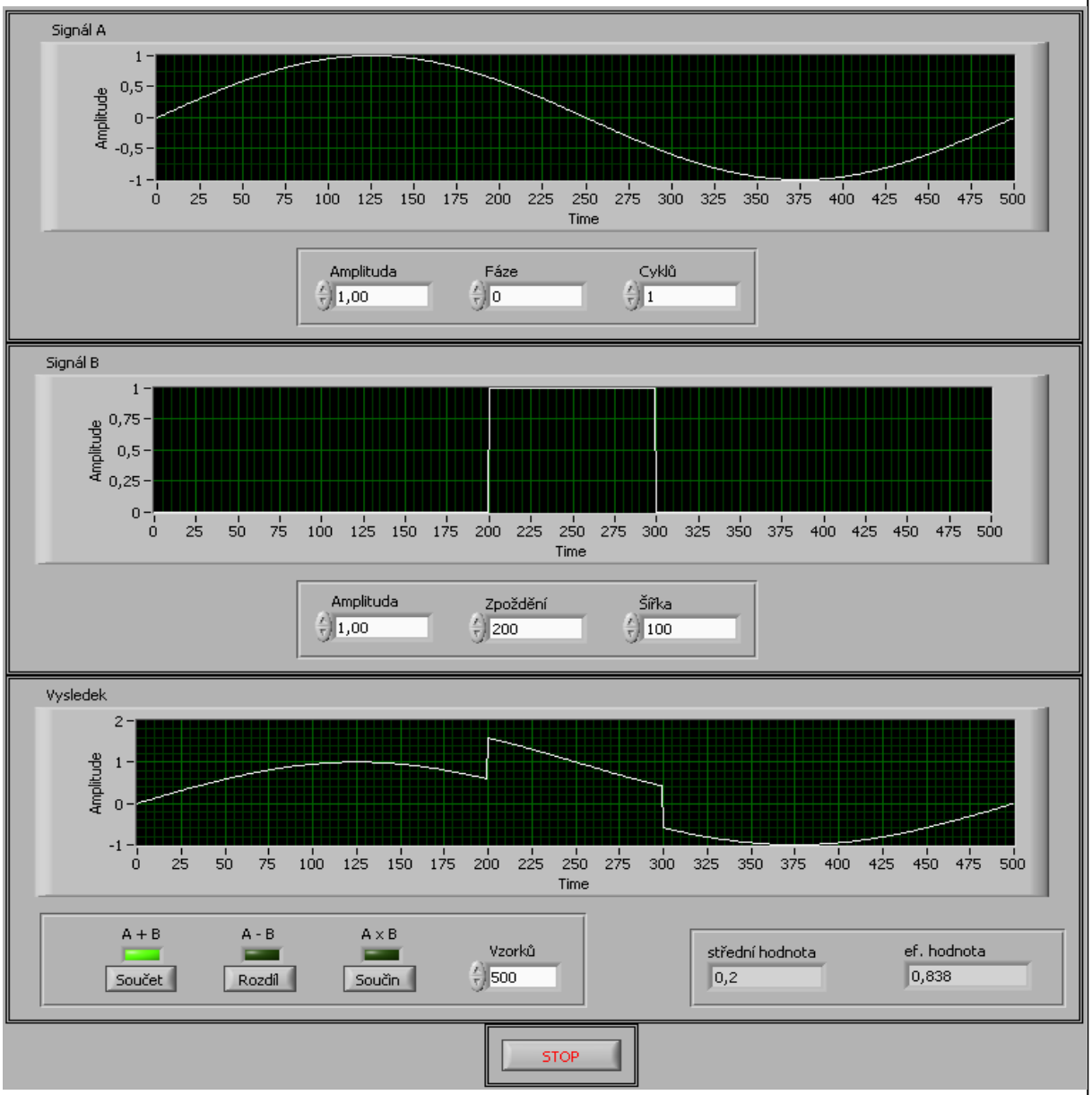
6) Opakujeme bod 6) pro zbylé operace (Multiply a Subtract).

7) Obdobná konstrukce je i u VI pro výpočet střední a efektivní hodnoty s tím rozdílem, že vstupem i výstupem nebude třída ale pole resp. 2 pole. A použijeme komponentu Basic Averaged DC-RMS.

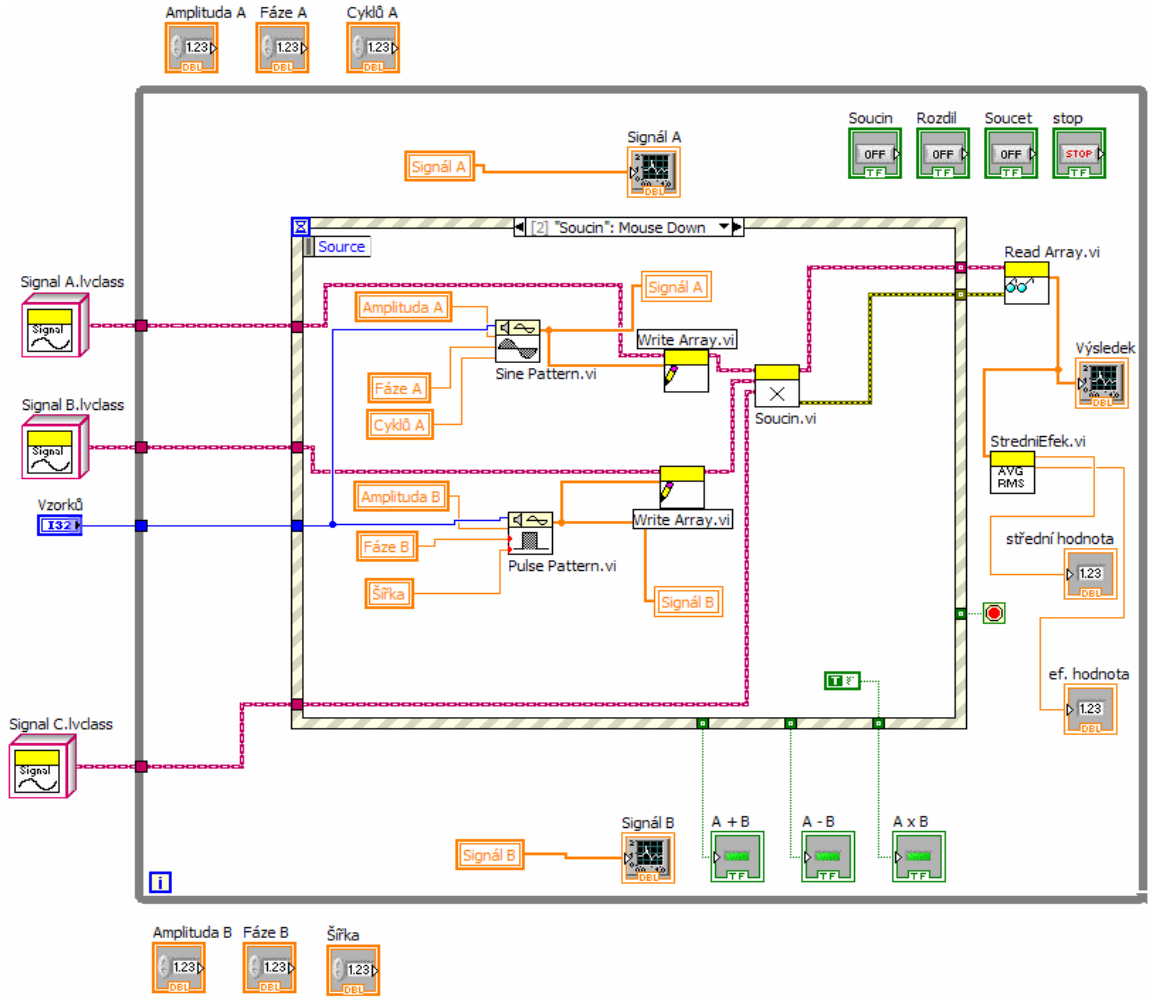
8) Co se týče třídy je vše hotovo a pustíme se vytvoření hlavního VI. V kontextové nabídce položky My Computer v Project Exploreru zvolíme New >> VI a pojmenujme ho, dle programátorských zvyků, Main.



9) Výsledné uživatelské rozhraní může vypadat např. takto:



10) Blokové schéma celé běží ve smyčce While, stisk některého z tlačítek (Součet, Součin, Rozdíl a Stop) je ošetřeno Case strukturou a generování signálů komponentami Sine Pattern a Pulse Pattern. Ukázku schématu vidíme na obrázku.



11.2.3 Zhodnocení

Tento program je spíše jednoduššího rázu. Pole obsahují jednorozměrná pole prvků navzorkovaných simulovaných signálů a s těmito poli se provádějí požadované operace. Aby se dal program použít k práci s reálnými signály, musely by se bloky simulující signály nahradit tzv. Tasky komunikujícími s měřicími kartami.