

Software Development with LabVIEW Object-Oriented Programming

Tomi Maila

NIDays 2008, Helsinki

May 7, 2008

expressionflow.com

Tomi Maila



Founder, Editor in Chief
ExpressionFlow



Research Scientist
University of Helsinki



LabVIEW Champions Program Member
National Instruments



Developer
OpenG.org

Contents

□ **LVOOP Basics**

□ Composition

□ Inheritance

□ Dynamic Dispatching

□ Recursion

LVOOP – LabVIEW Object-Oriented Programming

- A novel programming paradigm
- Integrates dataflow with object-oriented programming
- First introduced in LabVIEW 8.20
- Enhanced in LabVIEW 8.5

Tools

Line
Rectangle
Circle
Ellipse
Triangle

Color

RGB CMY

Red 0
Green 141
Blue 255



Undo

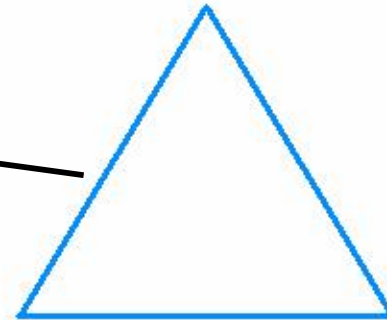
5 tools

drawing area

2 color models

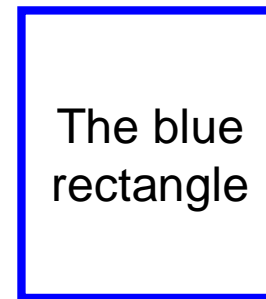
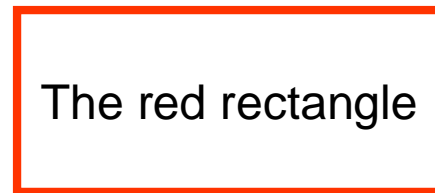
color indicator

undo button

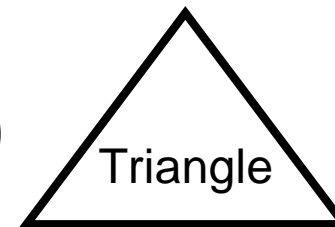
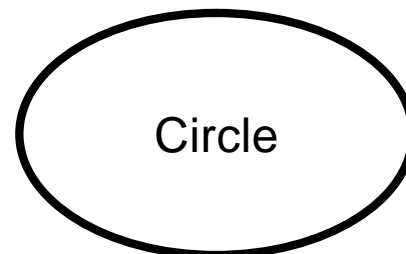
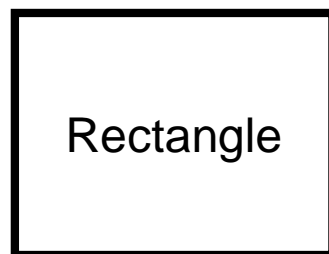


Objects & Classes

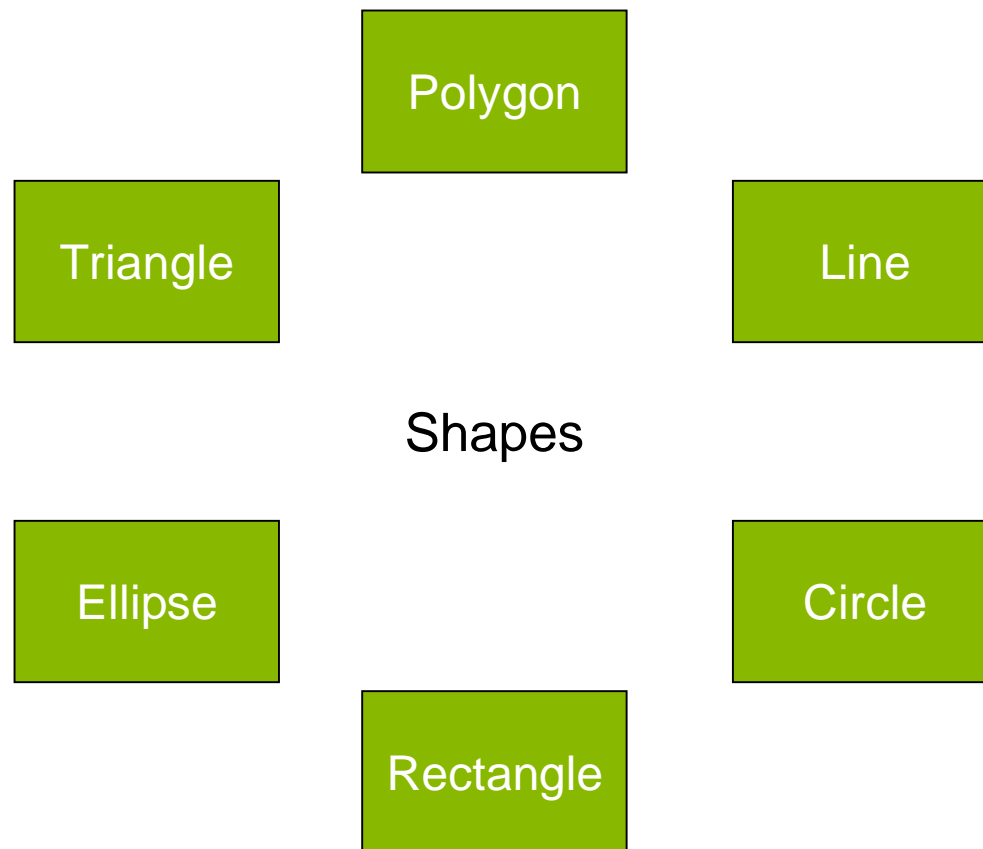
- Objects are individual items with particular properties such as color and size.



- Classes are collective items that describe the properties of particular kind of objects together

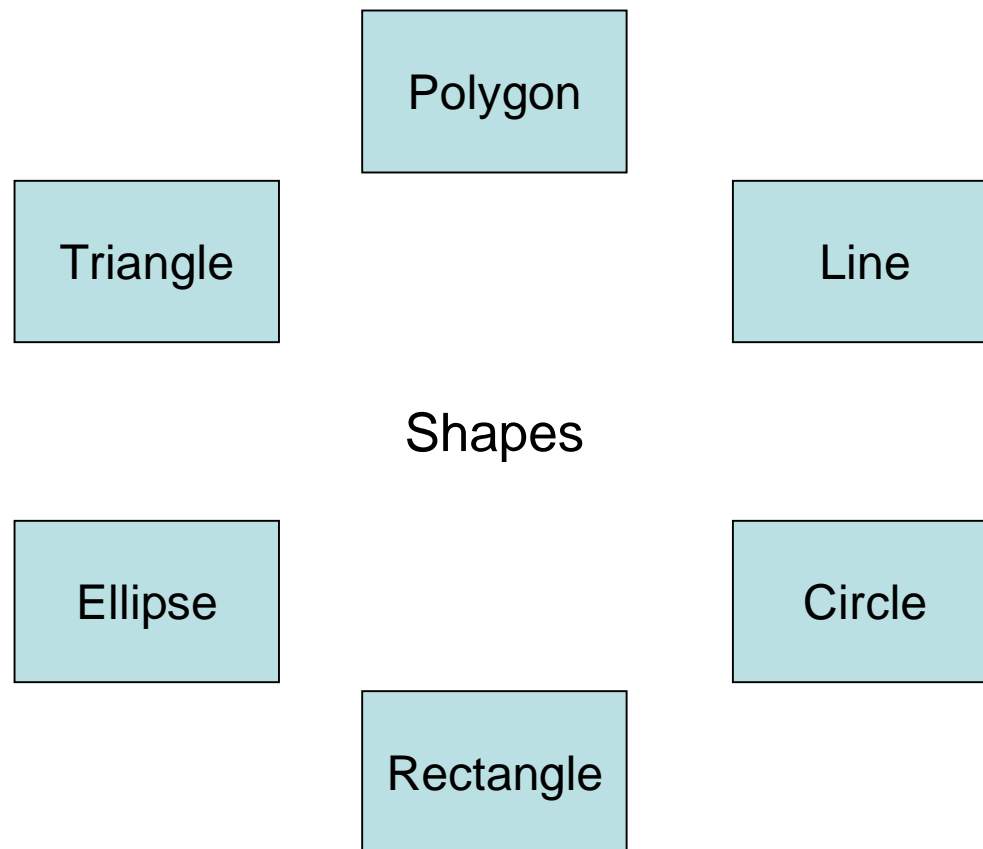


Real-World Shapes



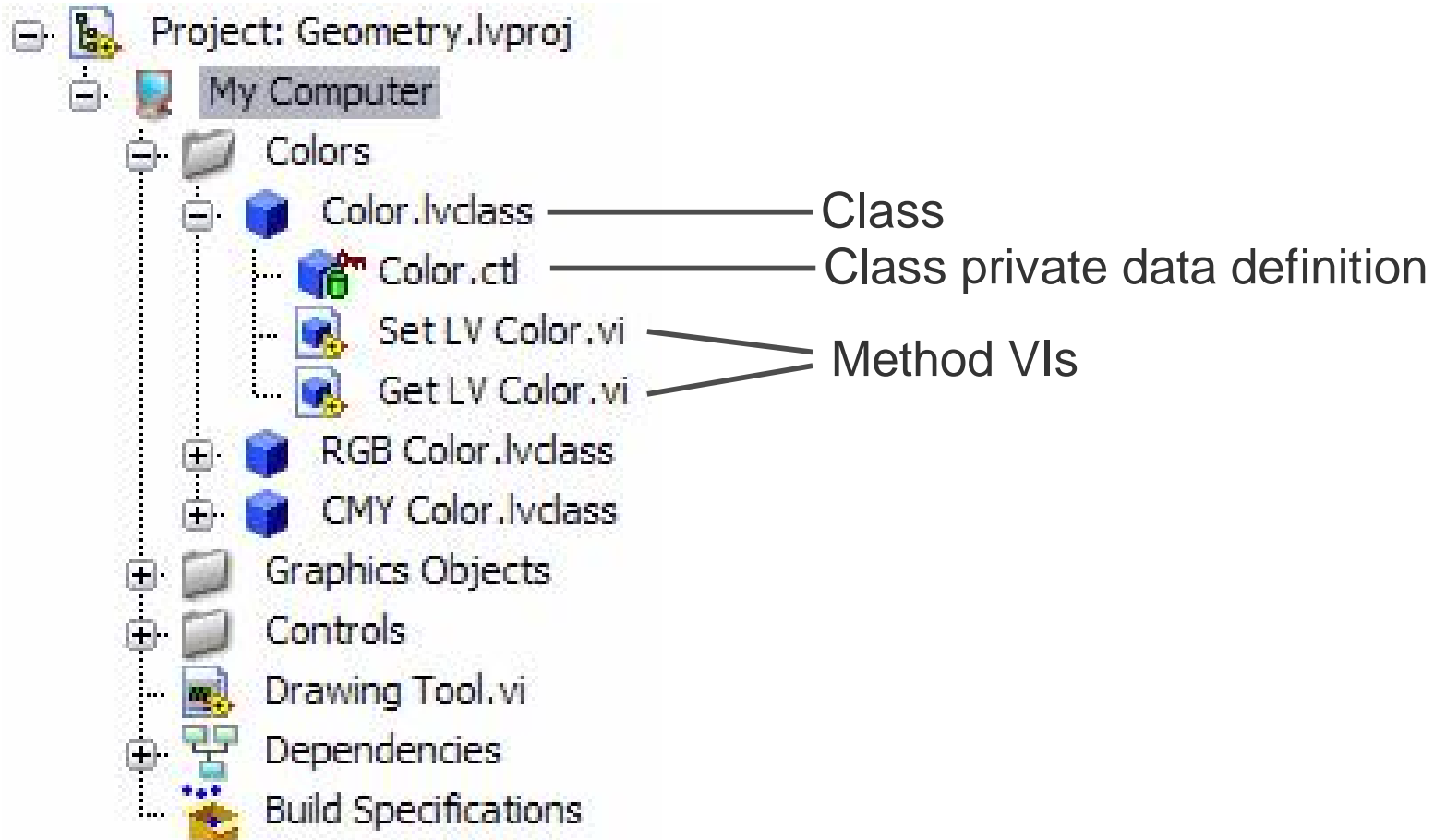
- There are relations between the shapes
- Objects of certain shape have properties such as size and color
- To draw a shape, some drawing instructions are needed

Shapes in LVOOP

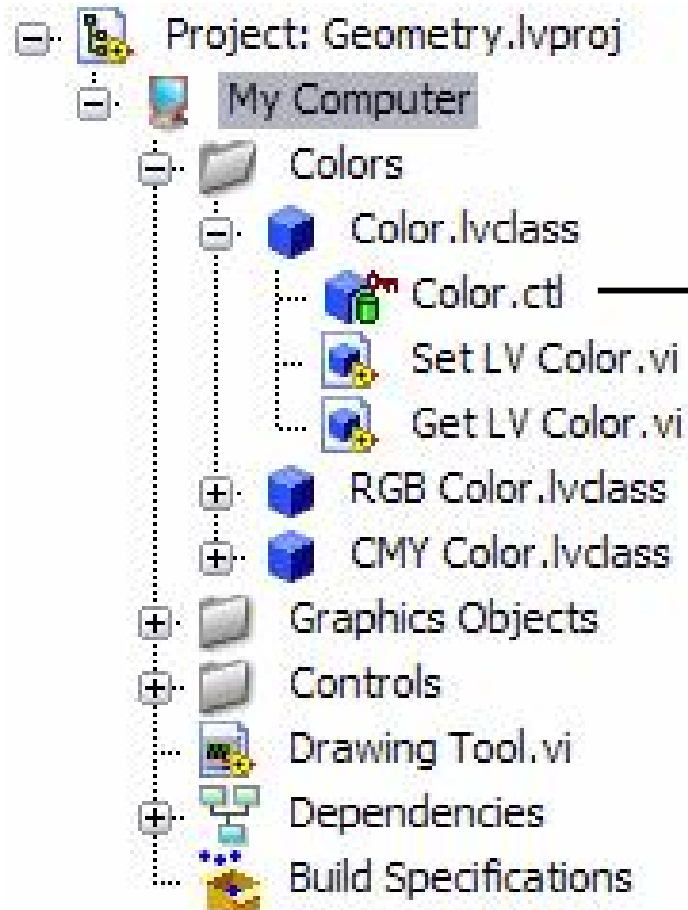


- There are relations between the shape classes
- Objects of shape classes have properties such as size and color
- To draw an object, some code is required

LVOOP Development with LabVIEW Project Explorer



What are LVOOP classes?



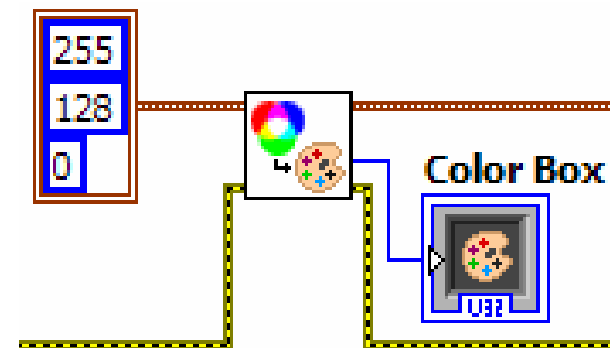
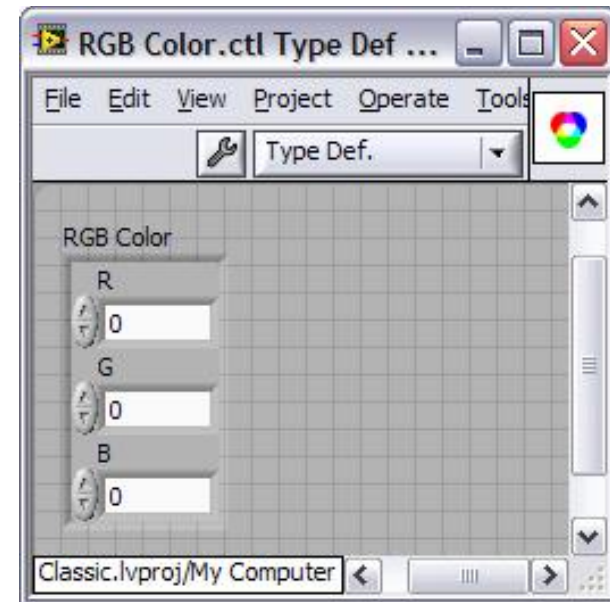
- Application building blocks
- Specify data type
- Specify functionality



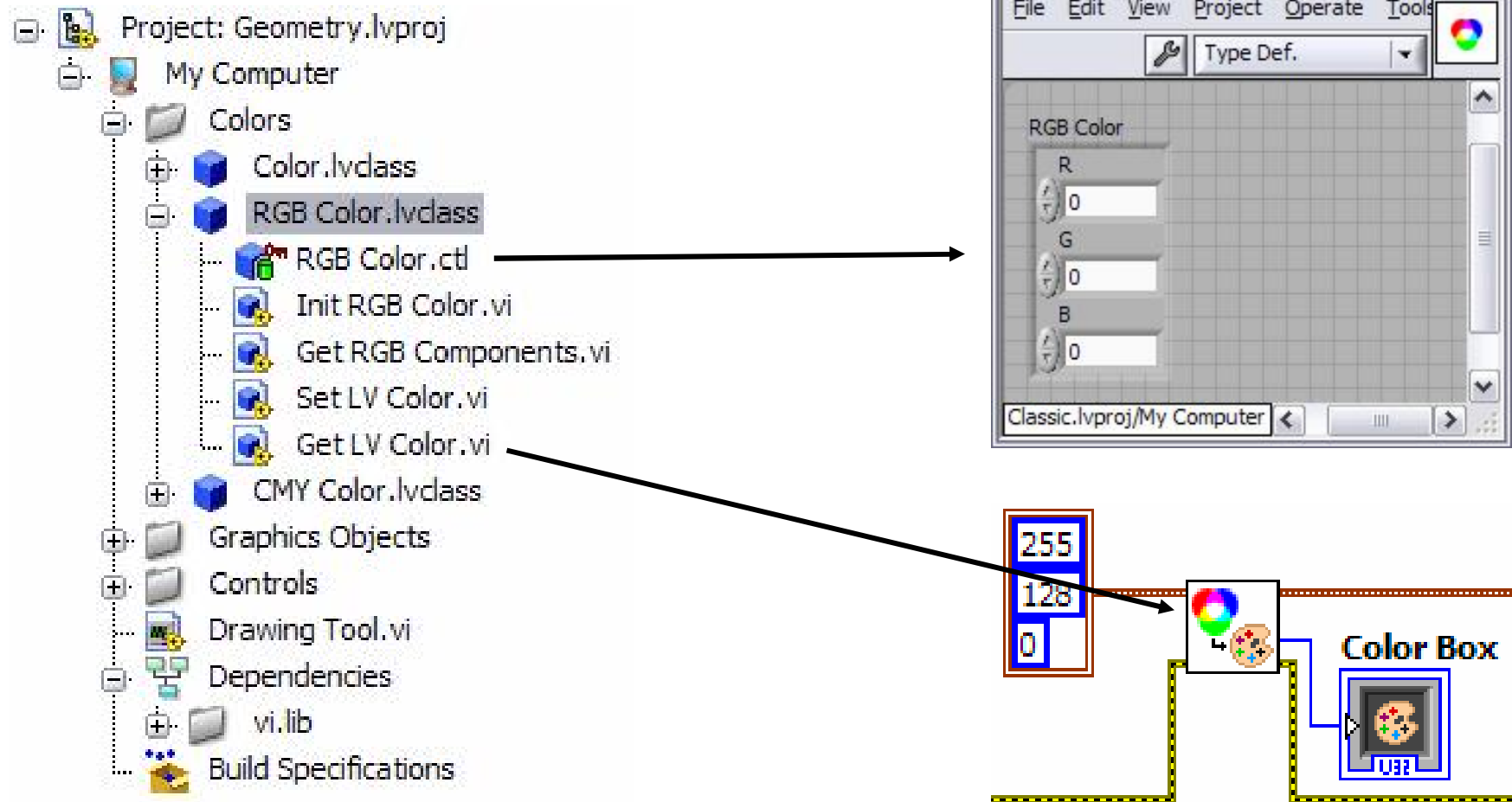
Abstraction in classical LabVIEW

- Data abstraction with Type Definitions

- Functionality abstraction with subVIs



Abstraction in LVOOP



What are LVOOP Objects?

- Class specifies the type of the wire
- Object is the state that flows along the wire

Class constant

Specifies the wire type & initial state



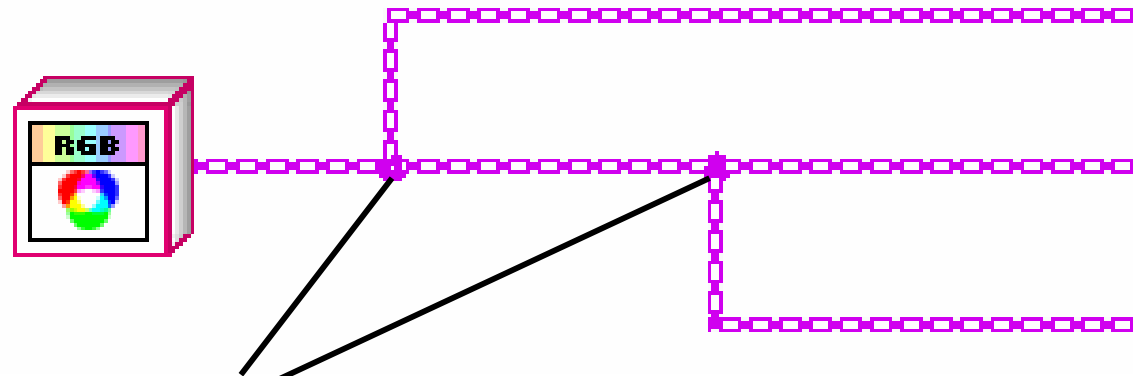
Object

State that flows along the wire



Dataflow Object-Oriented Programming

- ❑ Branching a wire creates a new object copy
- ❑ Each wire branch contains a separate object
- ❑ Differs from other programming languages like C# & Java
- ❑ LabVIEW doesn't have the concept of by-reference objects



New object copies created

Class Private Data Specify the Wire Content

The image shows a LabVIEW project explorer on the left and a 'Class Private Data' window on the right. The project explorer shows a project named 'Geometry.lvproj' with a folder 'Colors' containing several class files. The 'Class Private Data' window is titled 'RGB Color.ctl [Private ...]' and displays a 'Cluster of class private data' with three numeric controls labeled 'R', 'G', and 'B', each with a value of '0'. A 'Class icon' (a color wheel) is visible in the top right corner of the window. Two black lines with arrows point from the text labels below to the corresponding elements in the screenshot.

Project: Geometry.lvproj

- My Computer
 - Colors
 - Color.lvclass
 - RGB Color.lvclass
 - RGB Color.ctl
 - Init RGB Color.vi
 - Get RGB Components.vi
 - Set LV Color.vi
 - Get LV Color.vi
 - CMY Color.lvclass
 - Graphics Objects
 - Controls
 - Drawing Tool.vi
 - Dependencies
 - vi.lib
 - Build Specifications

RGB Color.ctl [Private ...]

File Edit View Project Operate Tool

Class Private Data

Cluster of class private data

R 0

G 0

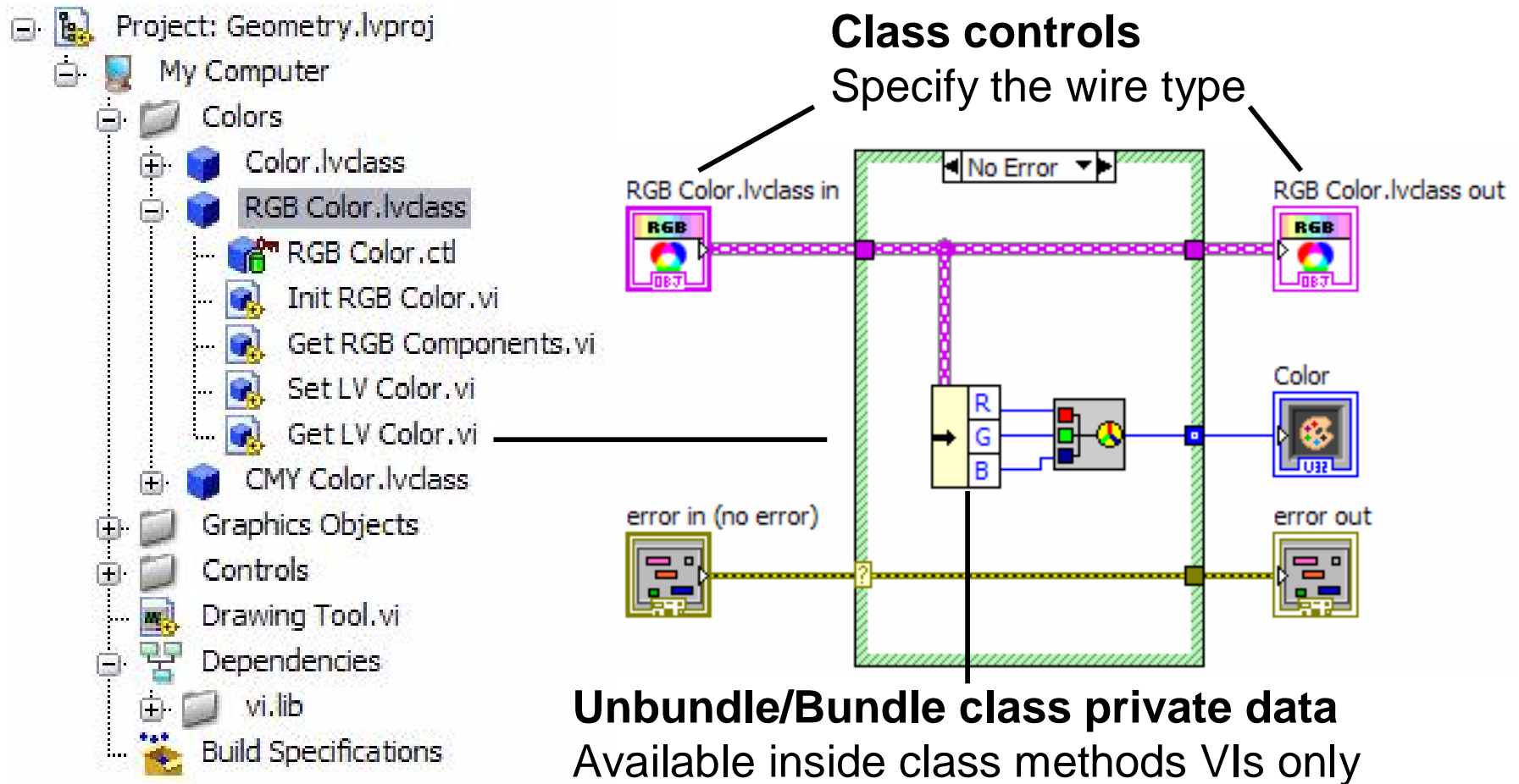
B 0

Geometry.lvproj/My Computer

Cluster of class private data

Class icon

Class Method VIs Modify the Object State

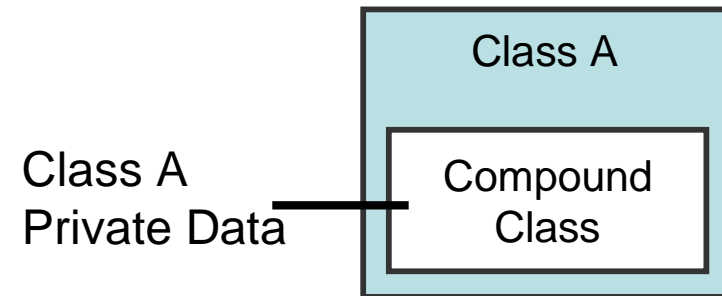


Contents

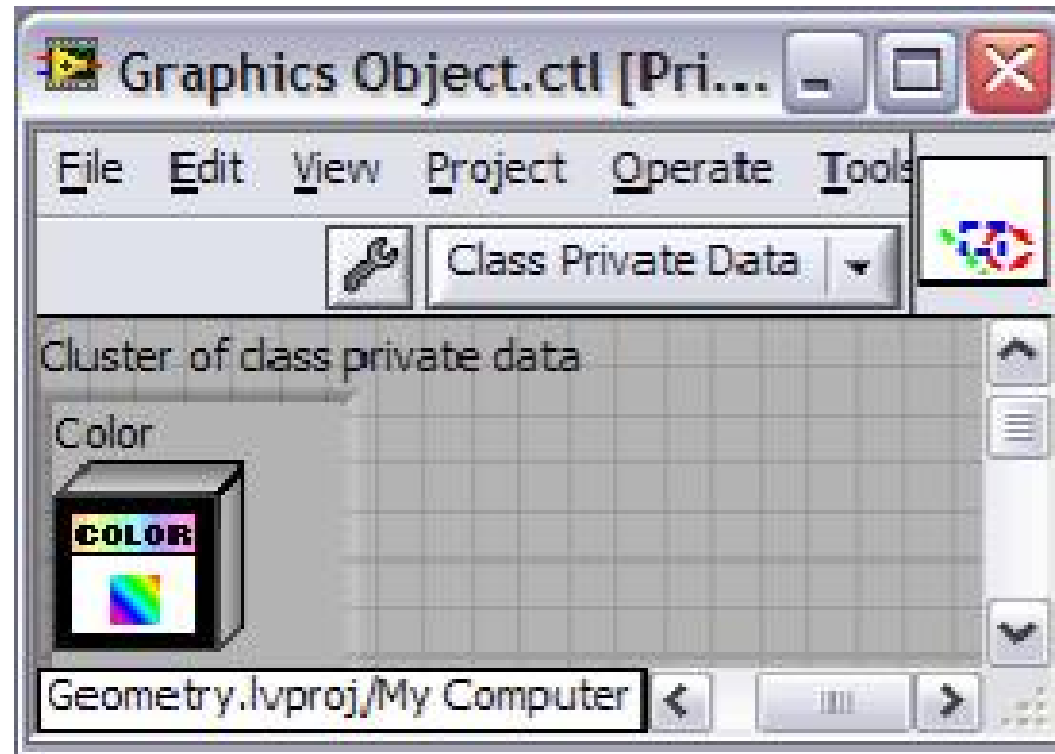
- LVOOP Basics
- **Composition**
- Inheritance
- Dynamic Dispatching
- Recursion

Composition

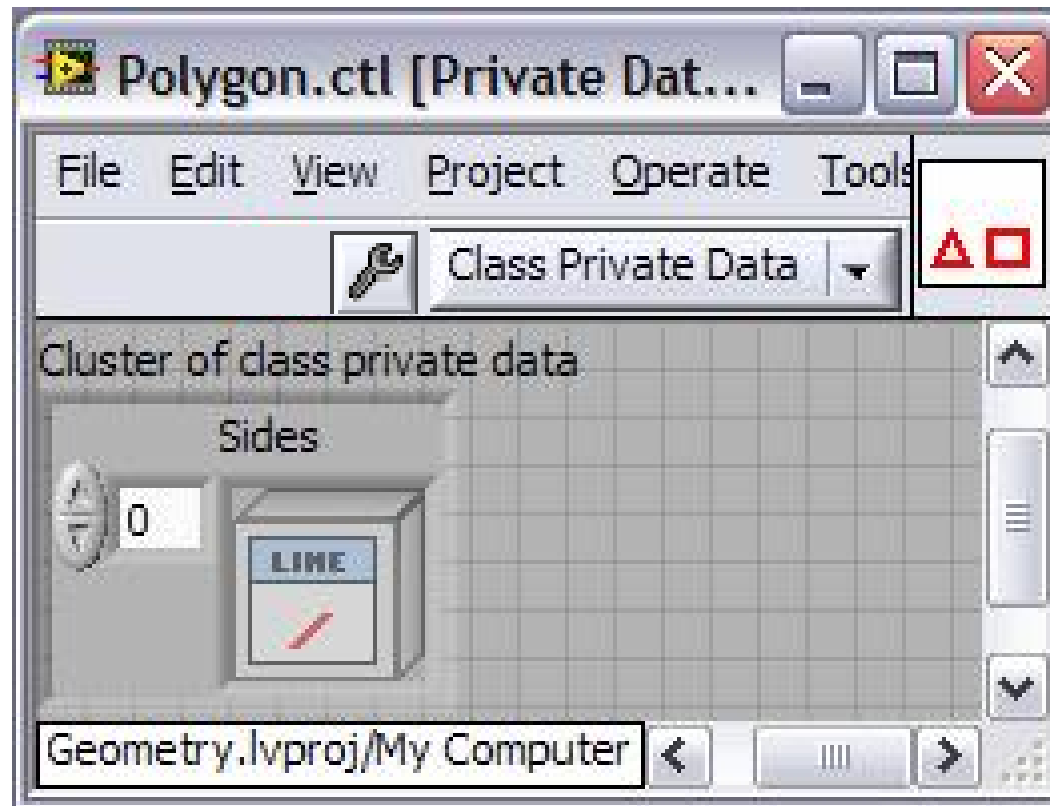
- ❑ Classes can be constructed by composing them of other classes
- ❑ Composition specified by placing compound class constant on the private data cluster of a class
- ❑ Compound class accessed through class methods



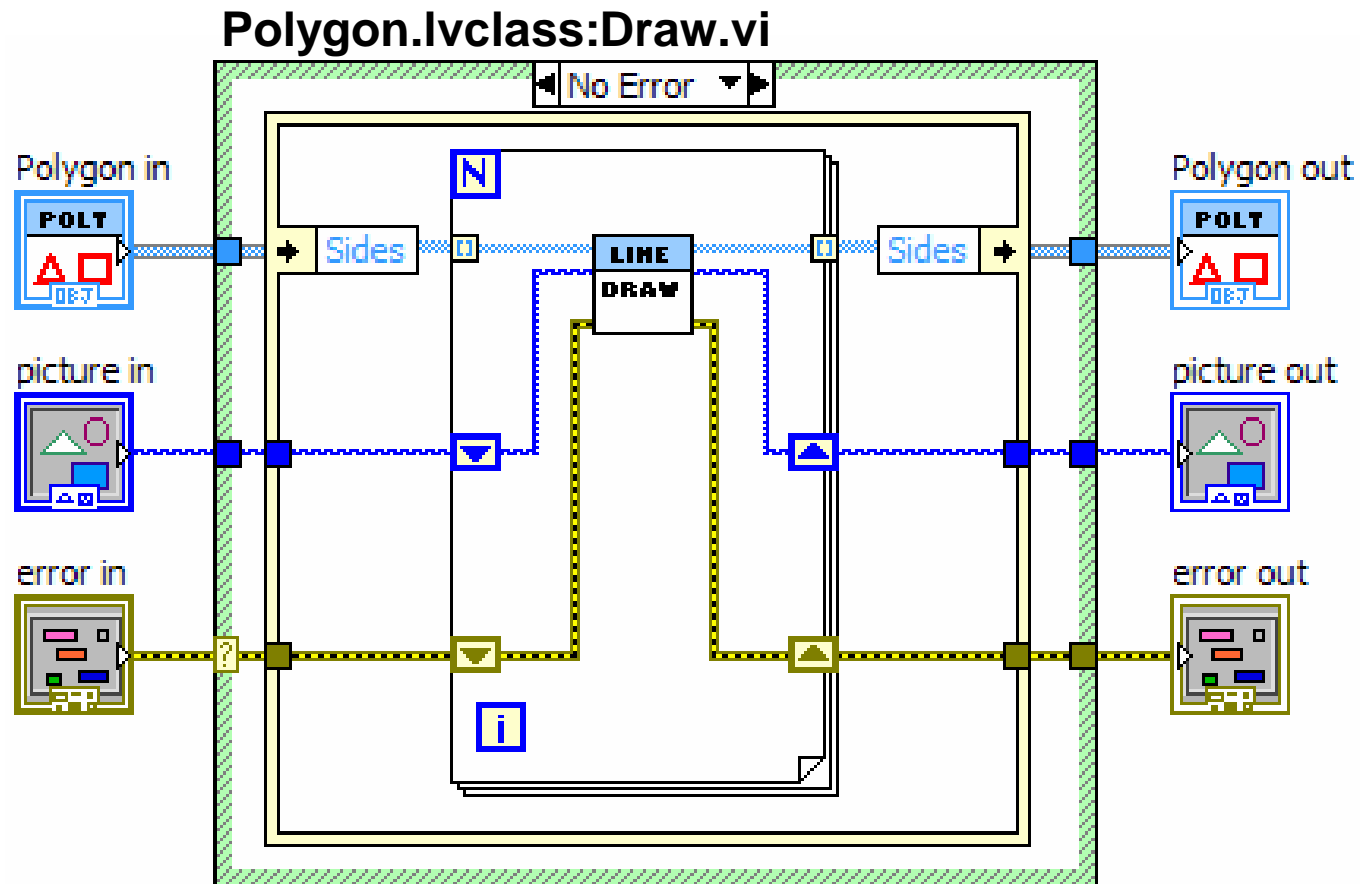
Graphics Object is Composed of Color



Polygon is Composed of an Array of Lines



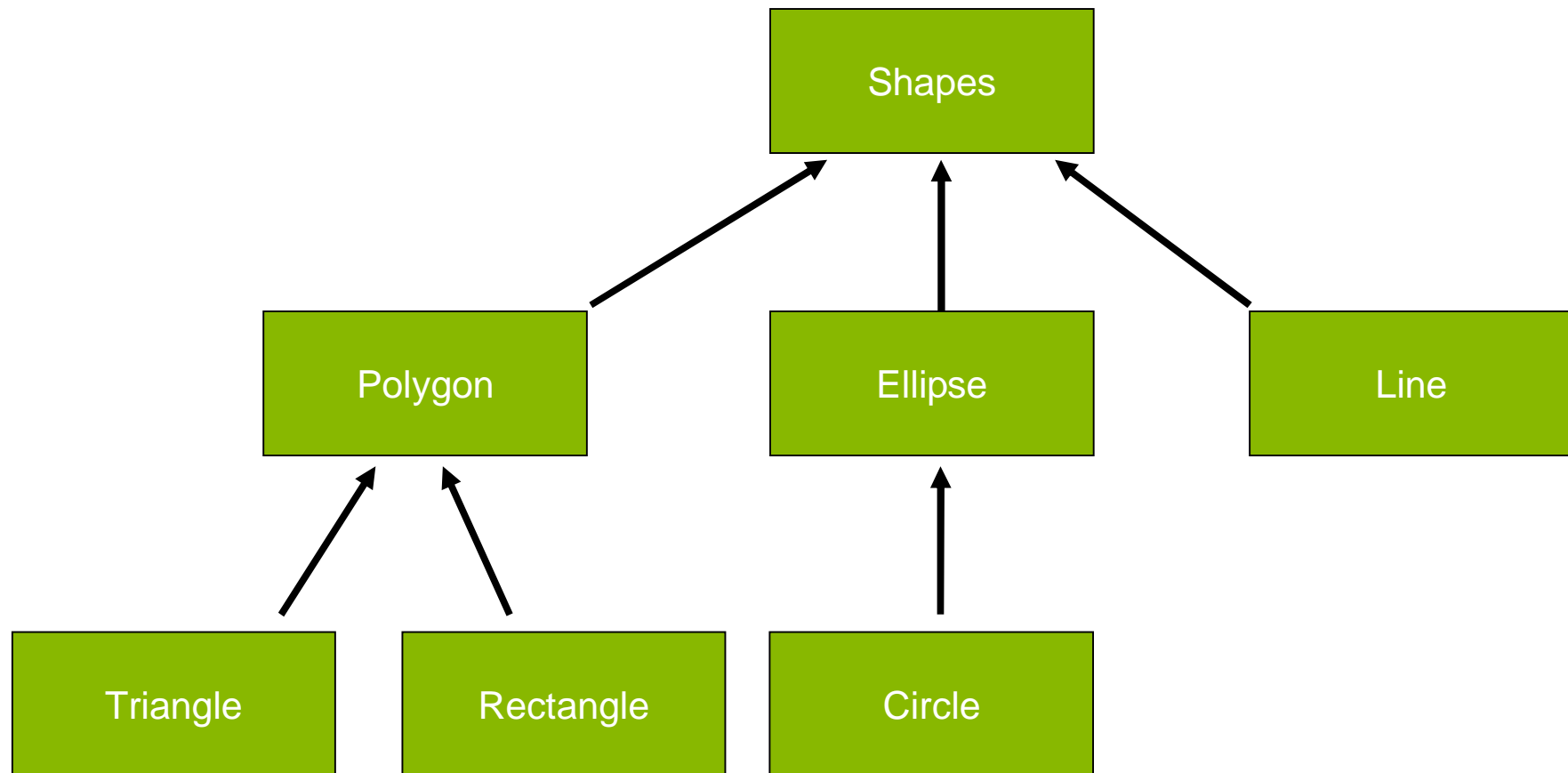
Task Delegation to Composite Objects



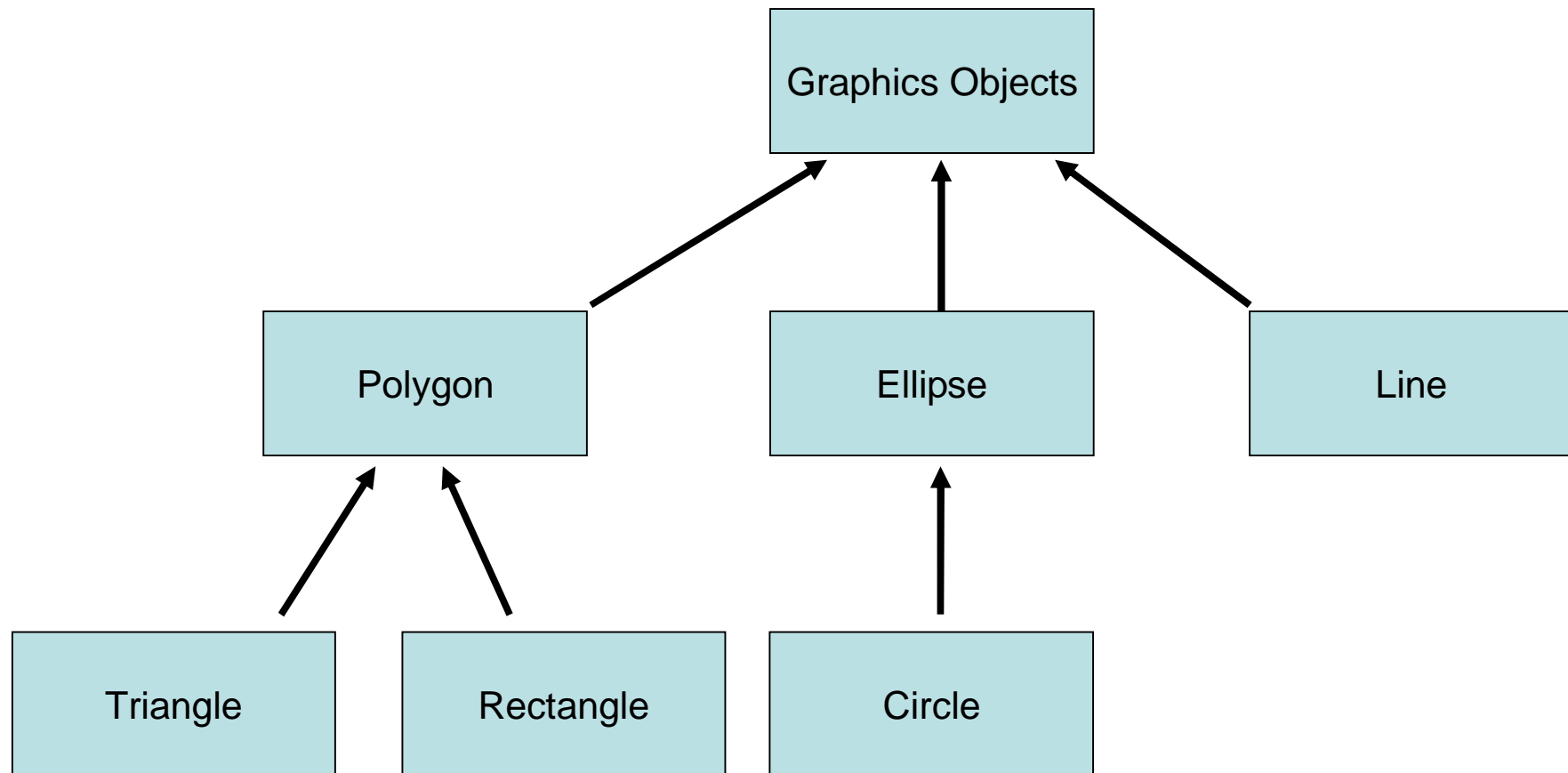
Contents

- LVOOP Basics
- Composition
- **Inheritance**
- Dynamic Dispatching
- Recursion

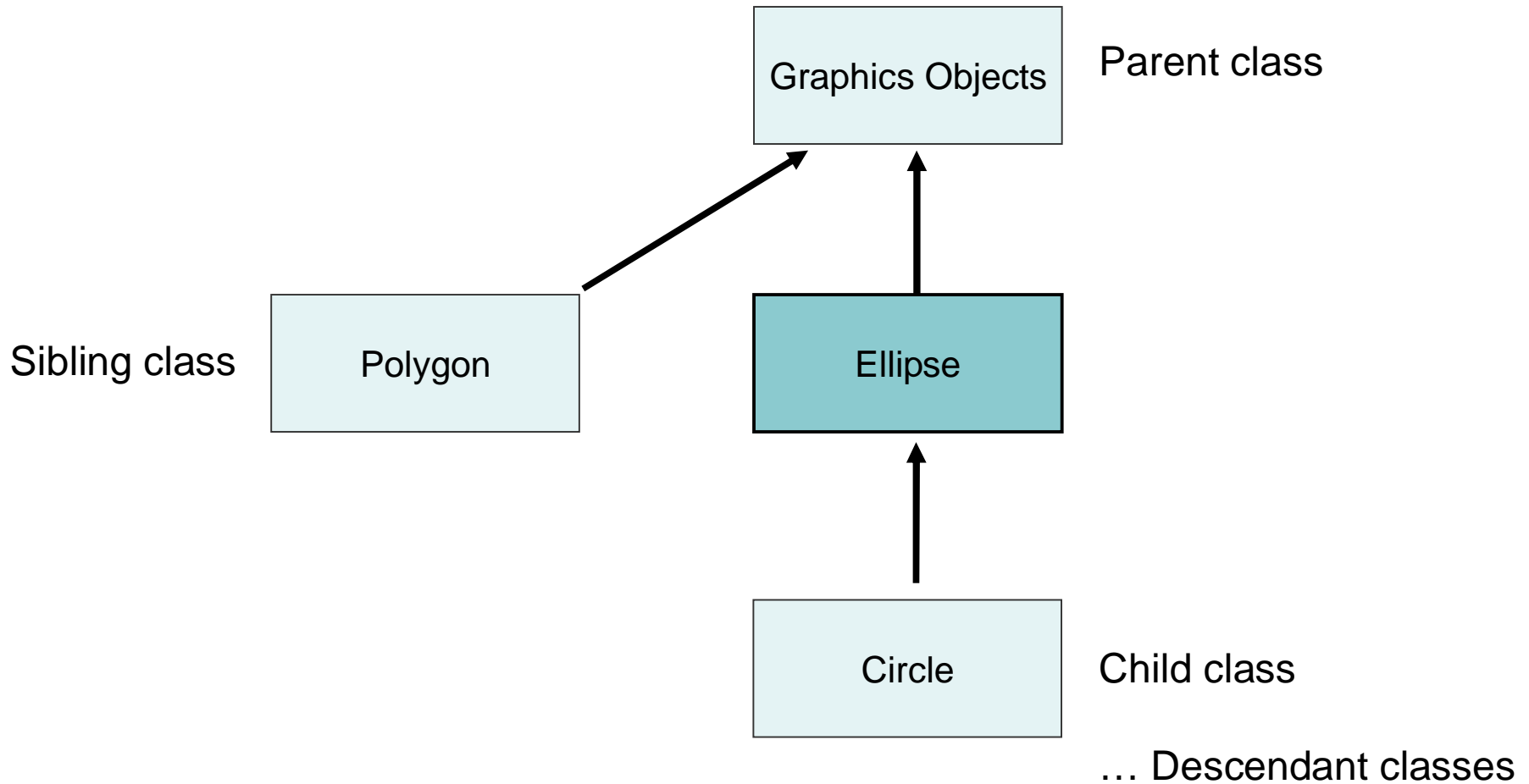
Conceptual Hierarchies of Real-World Objects



Inheritance & Class Hierarchies in LVOOP



Inheritance Vocabulary



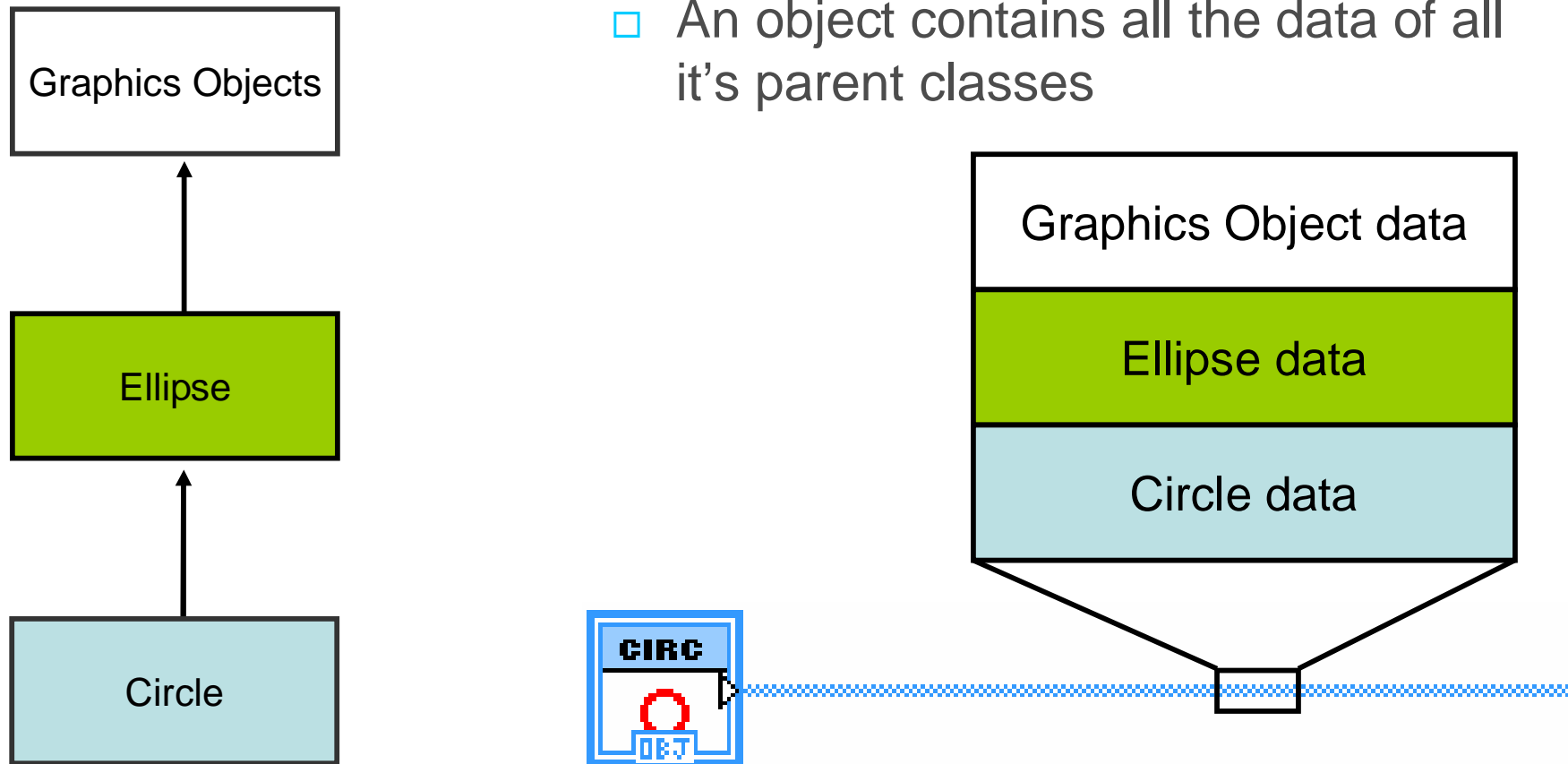
Why To Use Inheritance?

- Modularity
 - Inheritance allows us to divide our code into logical modules

- Code reuse
 - Code common to multiple descendant classes can be shared between the descendant classes through inheritance relations

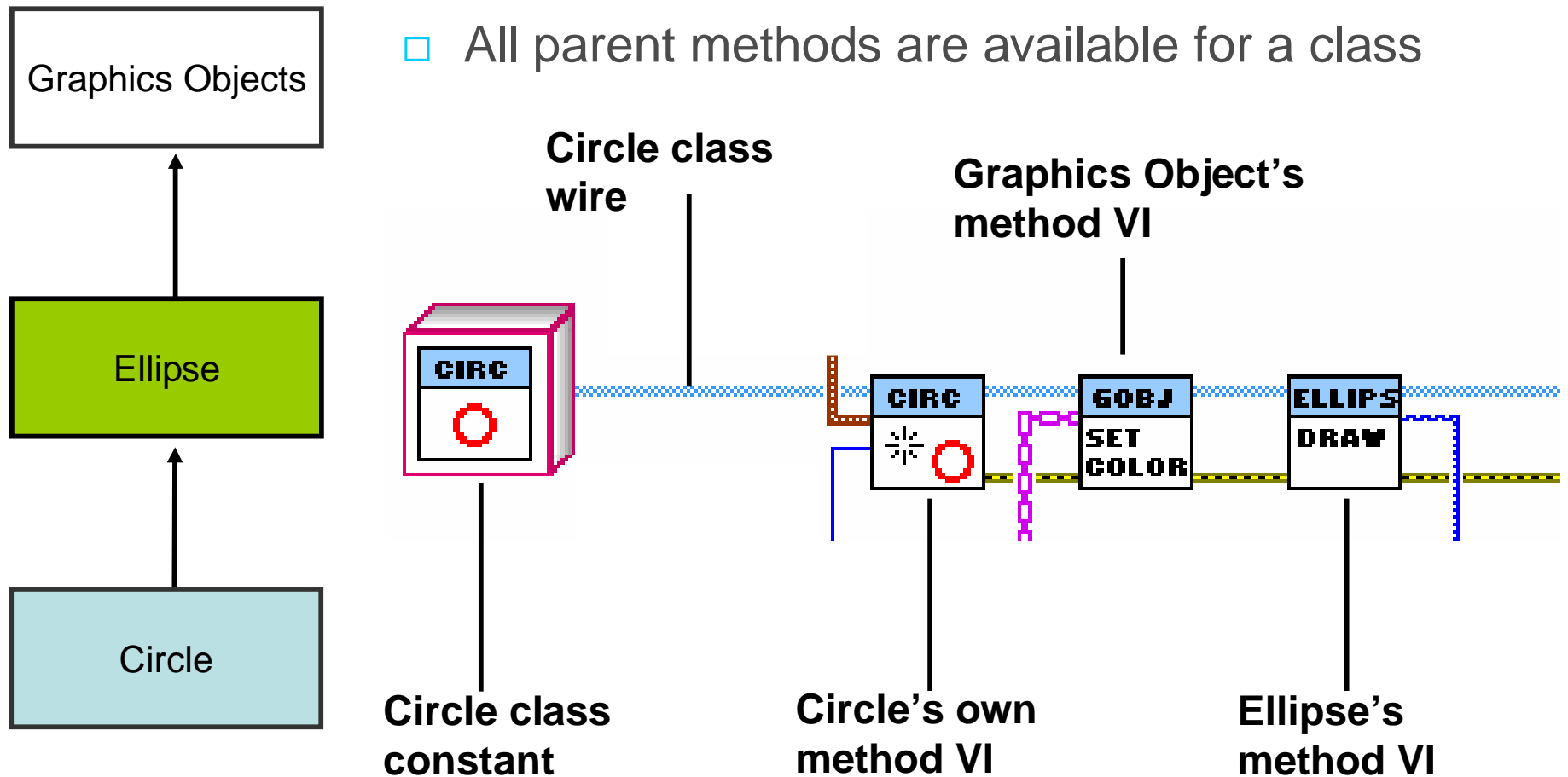
Data Inheritance

- An object contains all the data of all its parent classes

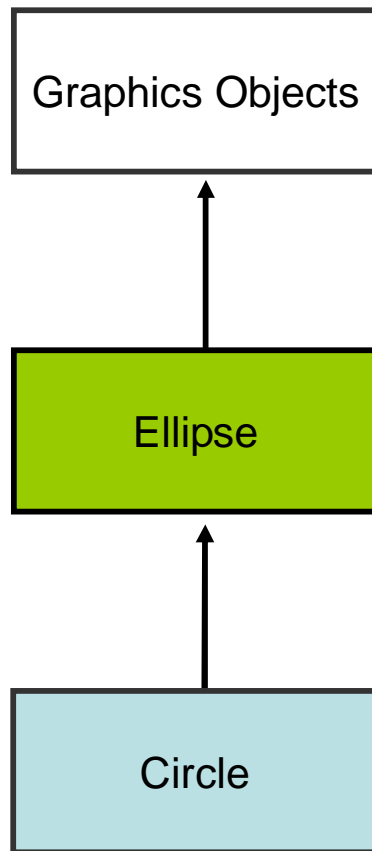


Method Inheritance

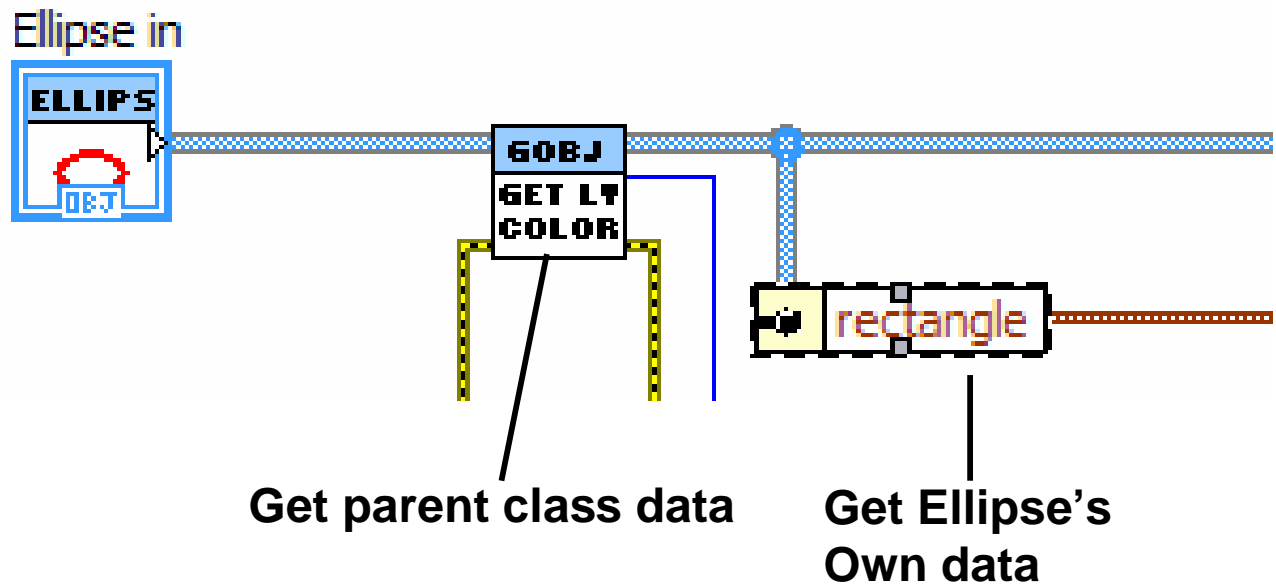
- All parent methods are available for a class



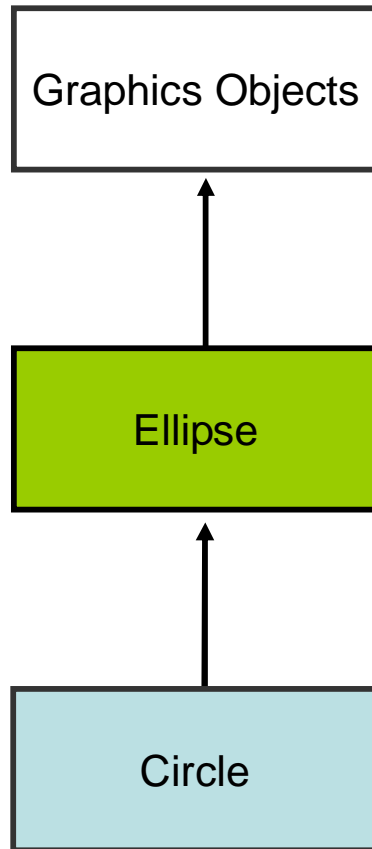
Data Privacy



- A class method VI can access only the data of it's own
- To access parent & ancestor data, ancestor method needs to be called



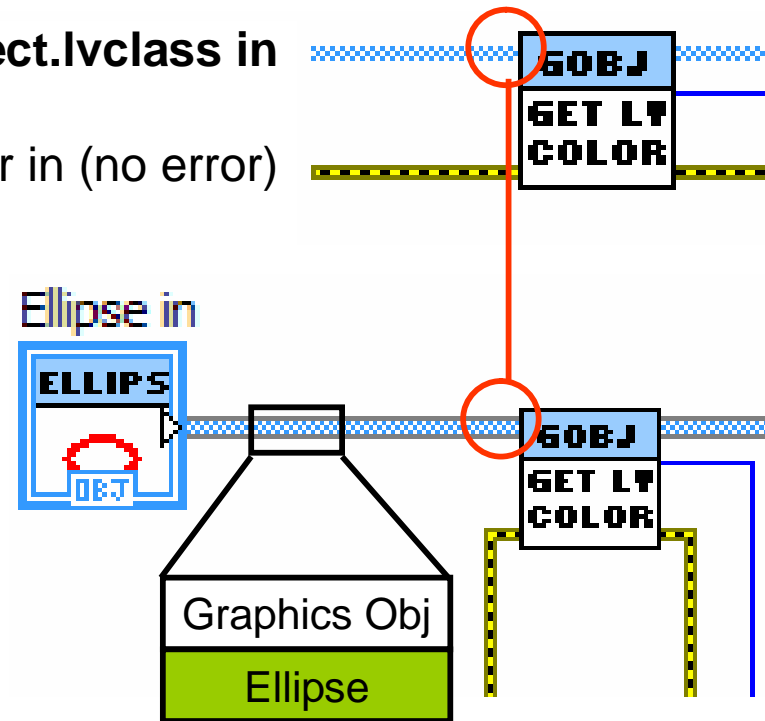
Class Terminals



- Class terminals accept classes of all child types

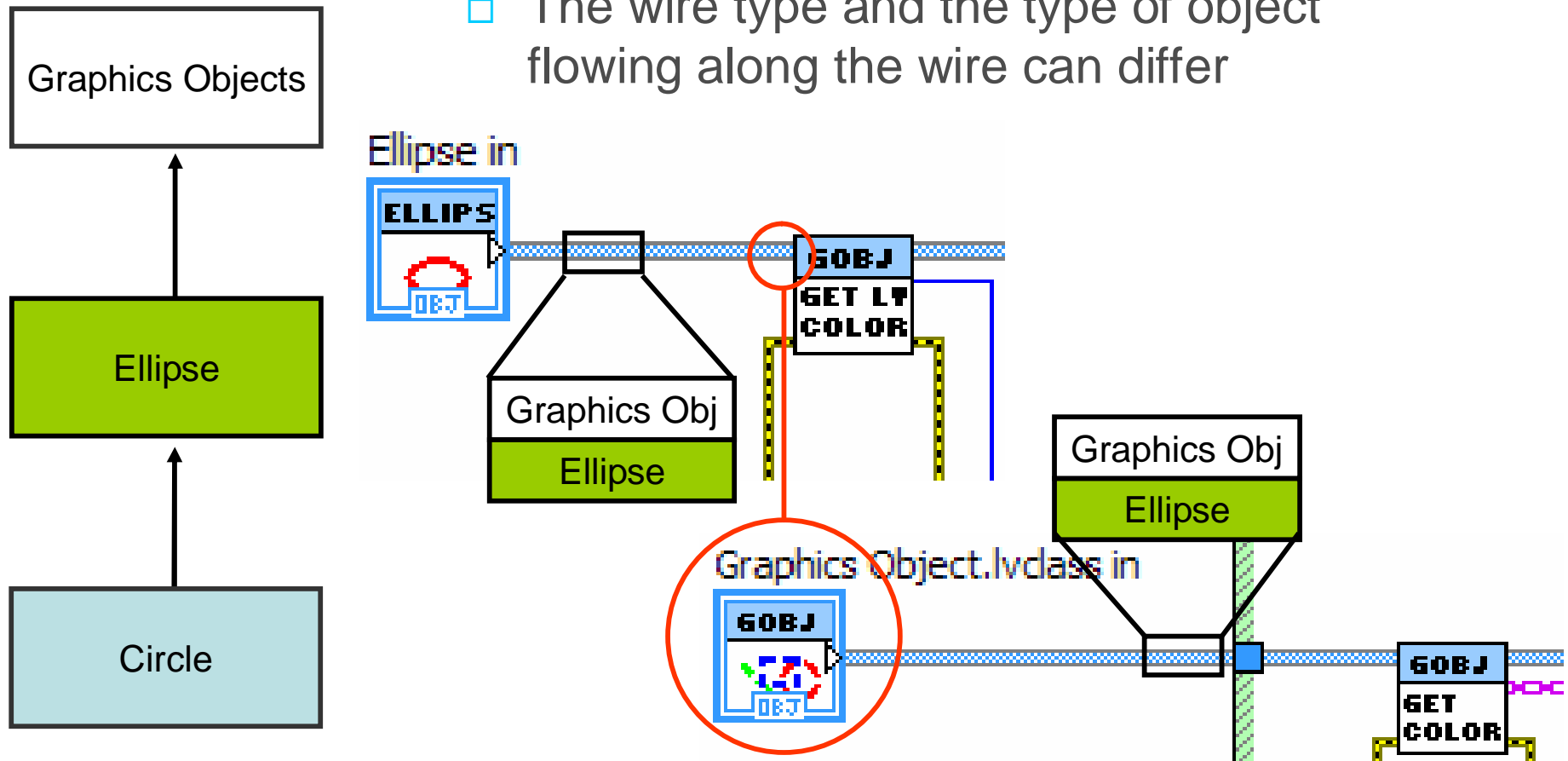
Graphics Object.lvclass in

Error in (no error)



Runtime Type

- The wire type and the type of object flowing along the wire can differ



Choosing Between Composition and Inheritance

- Composition when object naturally is constructed from parts
 - Polygon is constructed of multiple lines forming the sides

- Inheritance only when objects really is of parent type as well
 - Rectangle is a polygon, hence rectangle can inherit from polygon
 - Polygon is not a line, hence polygon cannot inherit from line

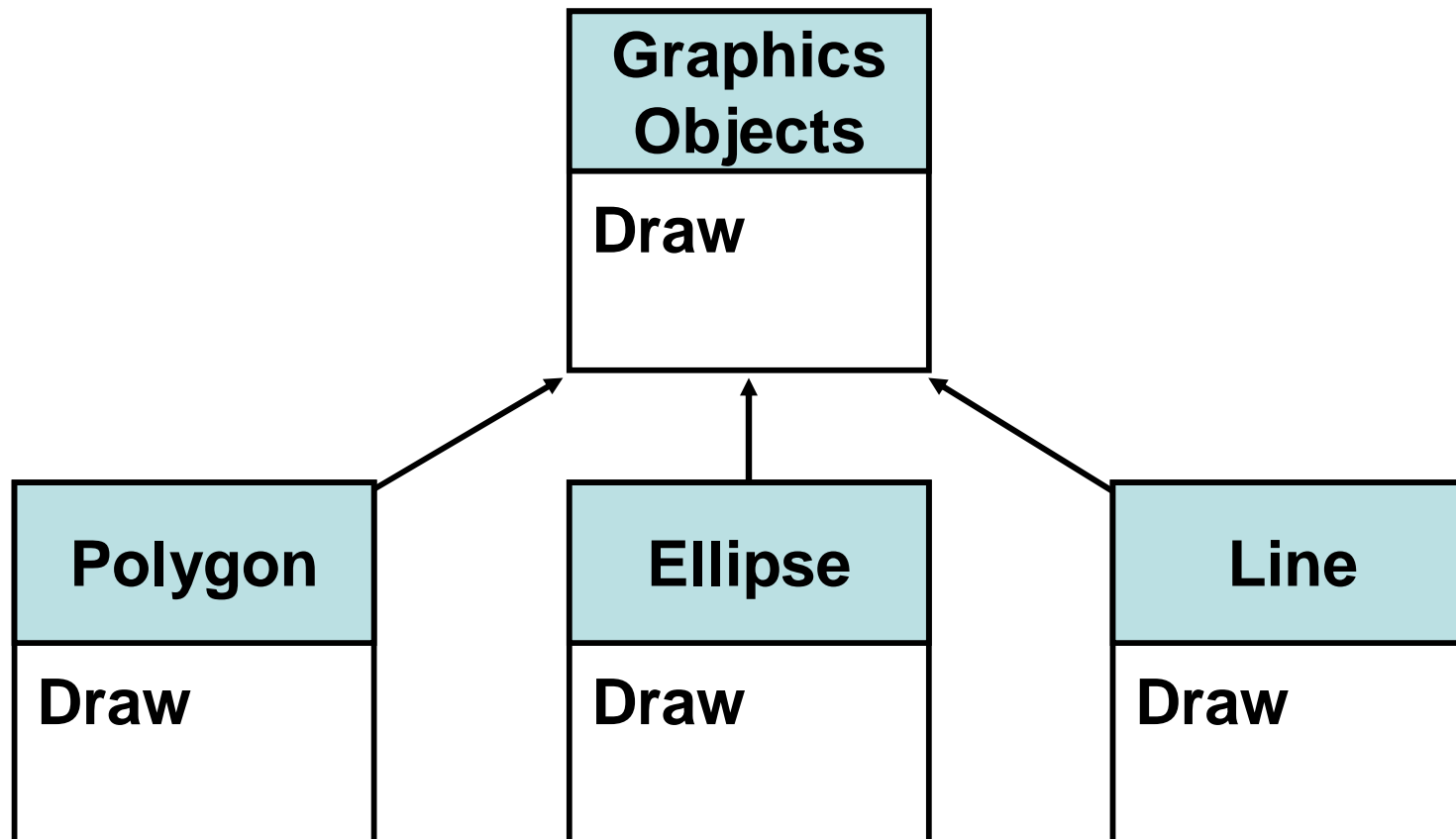
Contents

- LVOOP Basics
- Composition
- Inheritance
- **Dynamic Dispatching**
- Recursion

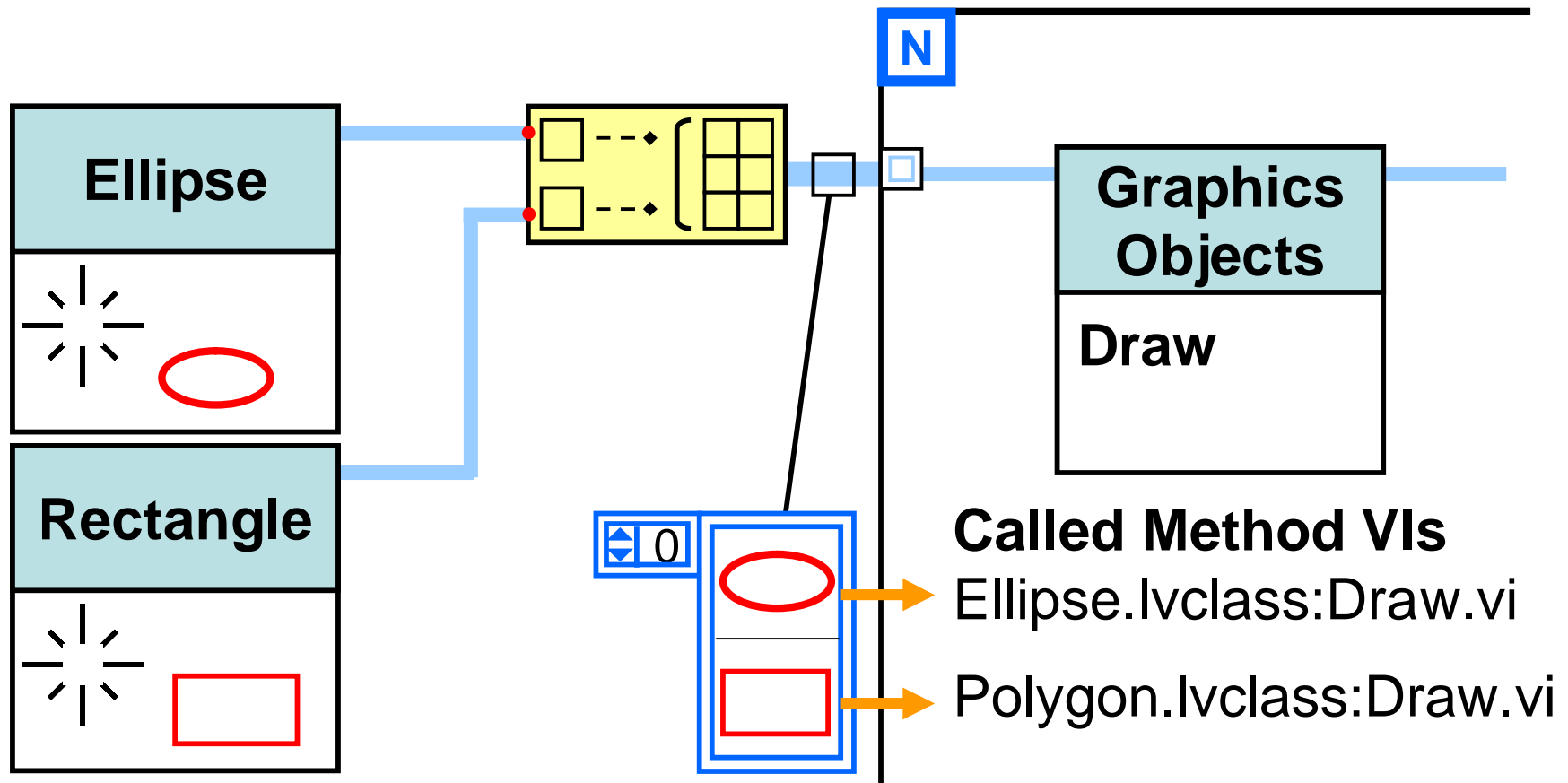
Dynamic Dispatch Method VIs

- Ability to change parent class method VI functionality
- Child class can override parent's dynamic dispatch method VI
- LabVIEW decides at runtime which dynamic dispatch method VI is actually called
- The called method is from the class specified by the object runtime type

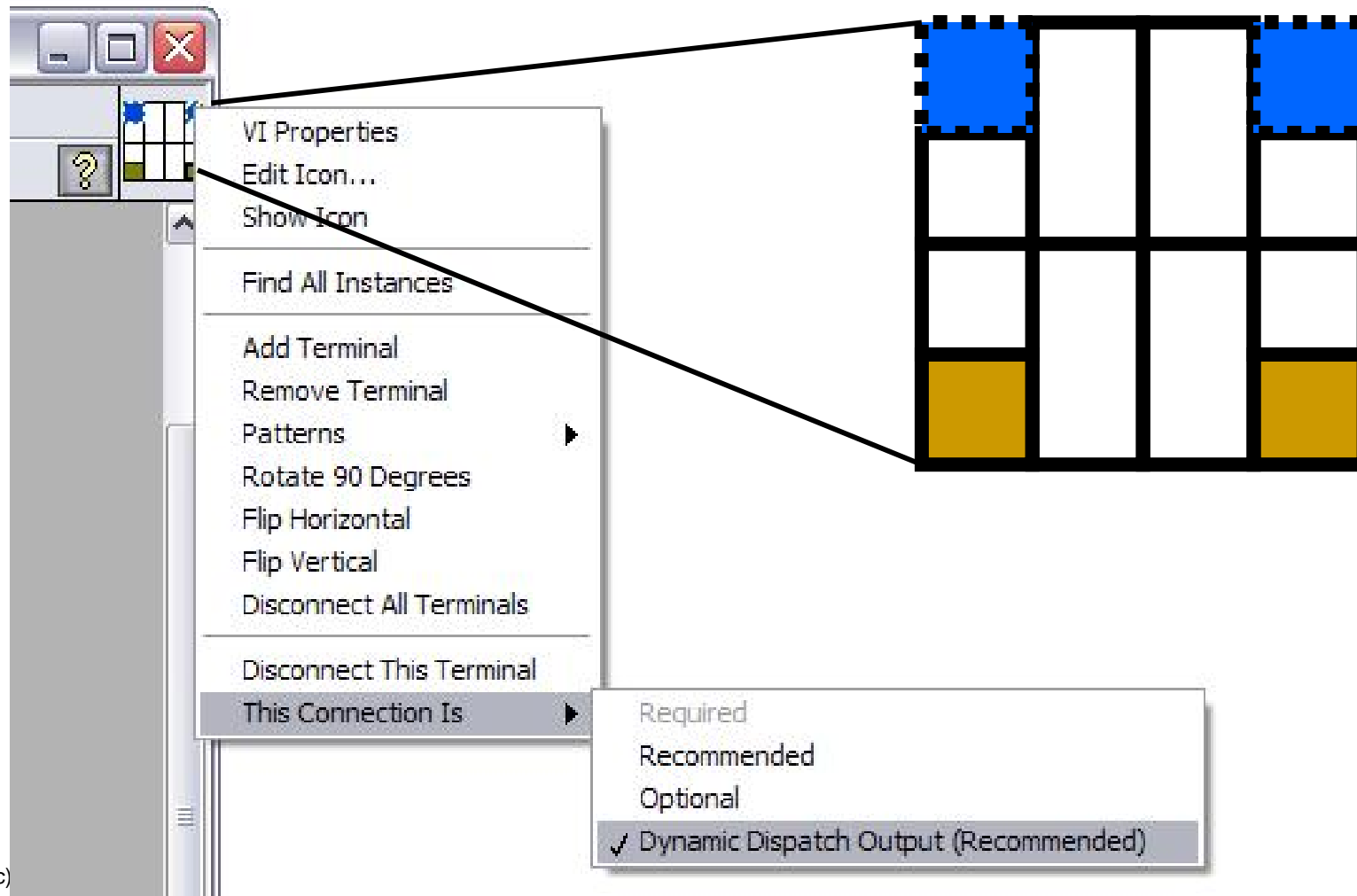
Overriding Functionality with Dynamic Method VIs



Dynamic Dispatching at Runtime



Dynamic dispatch terminals



Tools

Line
Rectangle
Circle
Ellipse
Triangle

Color

RGB CMY

Red 0
Green 141
Blue 255

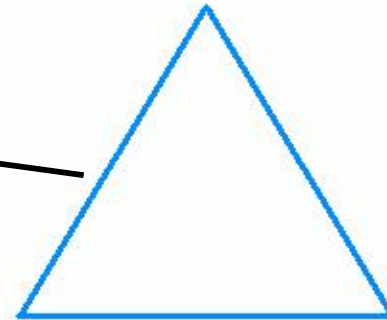


Undo

5 tools



drawing area



2 color models

color indicator

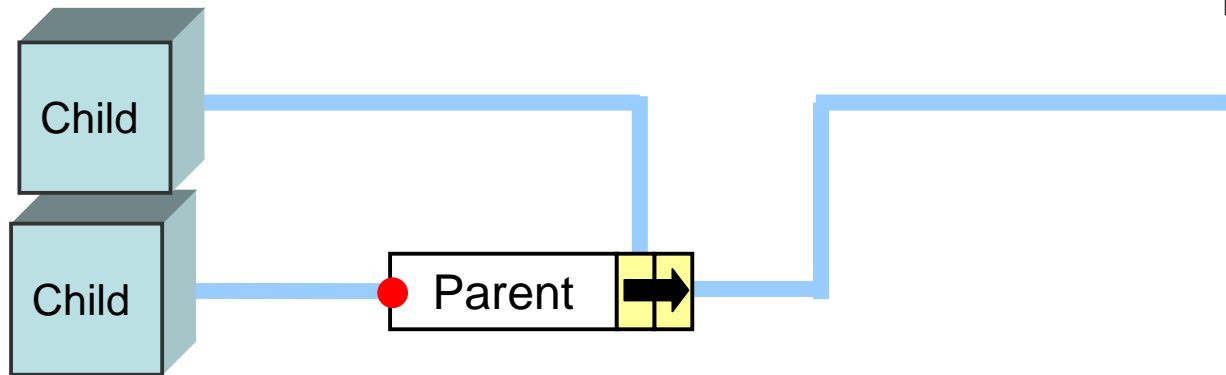
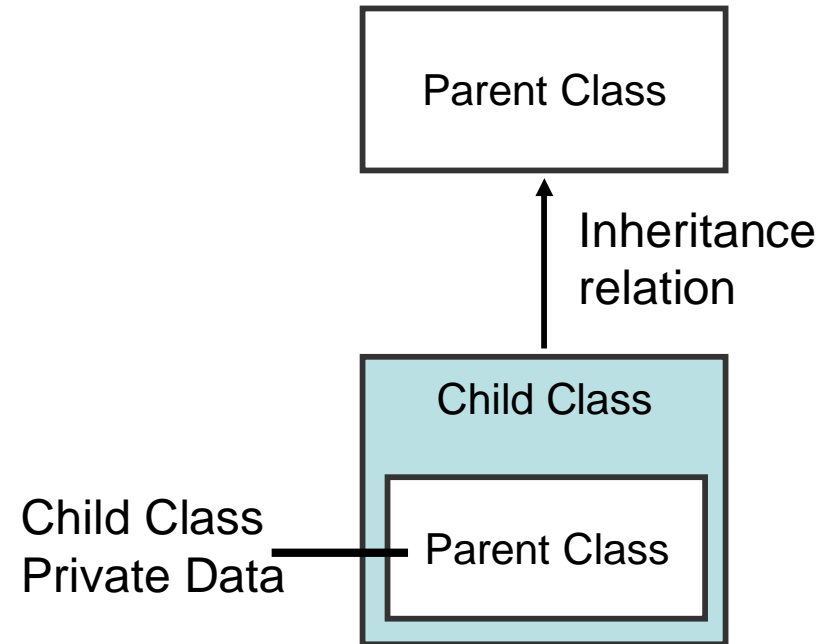
undo button

Contents

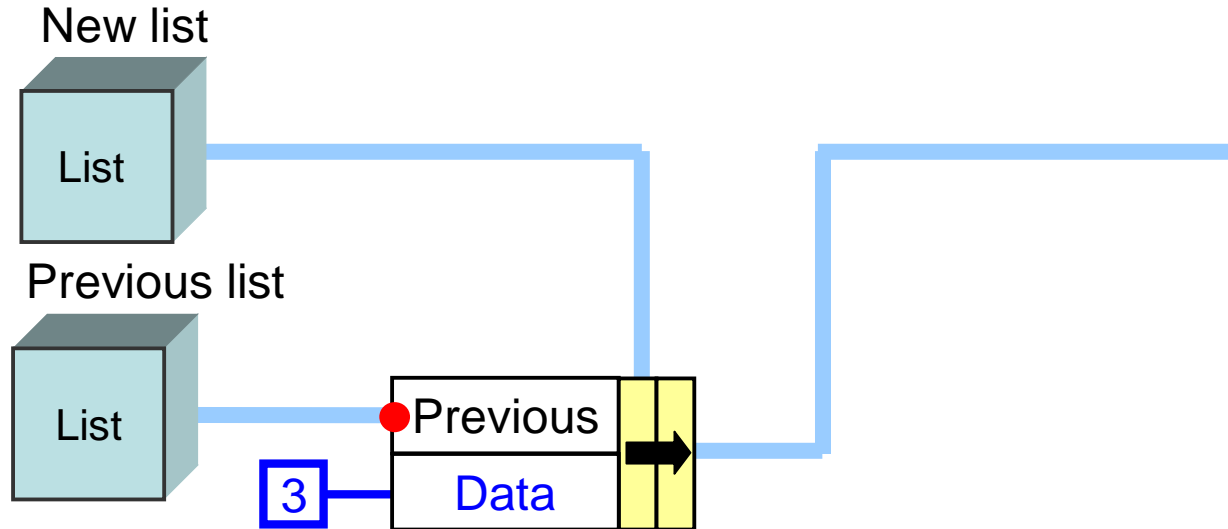
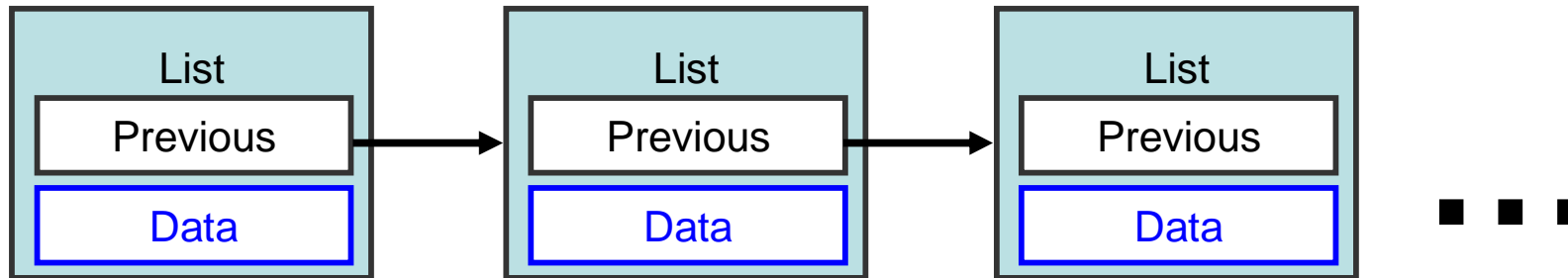
- LVOOP Basics
- Composition
- Inheritance
- Dynamic Dispatching
- **Recursion**

Recursive Composition

- Class private data can have elements of its parent type
- Parent type elements can have a value of the class itself

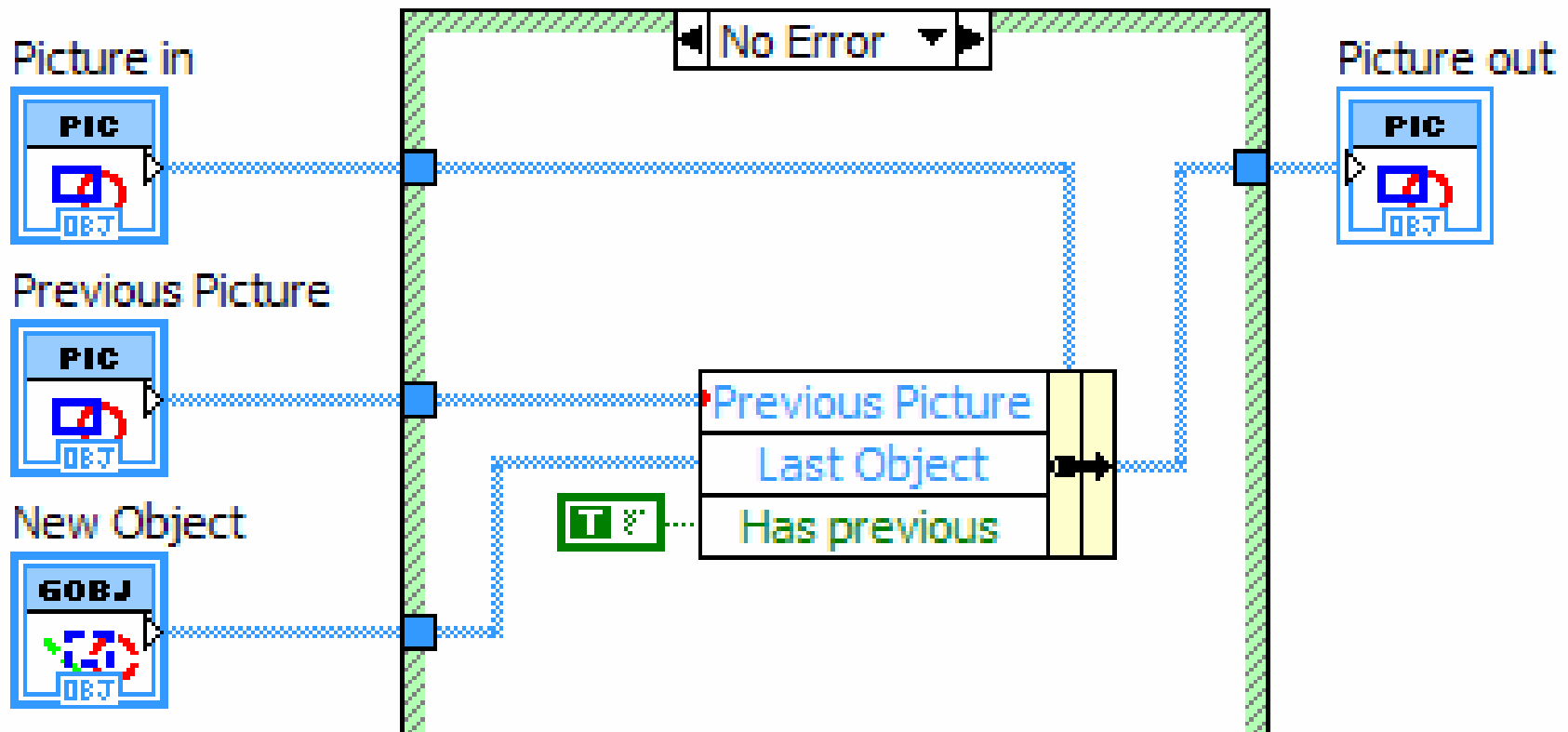


Linked List with Recursive Composition



Implementing Picture Class as a Linked List

Picture.lvclass:Init Picture.vi

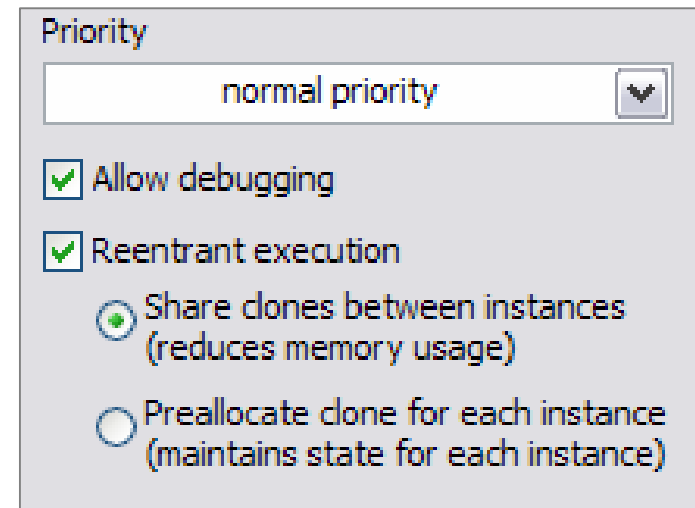


Recursive Calls

- Recursive method VI calls introduced in LabVIEW 8.5

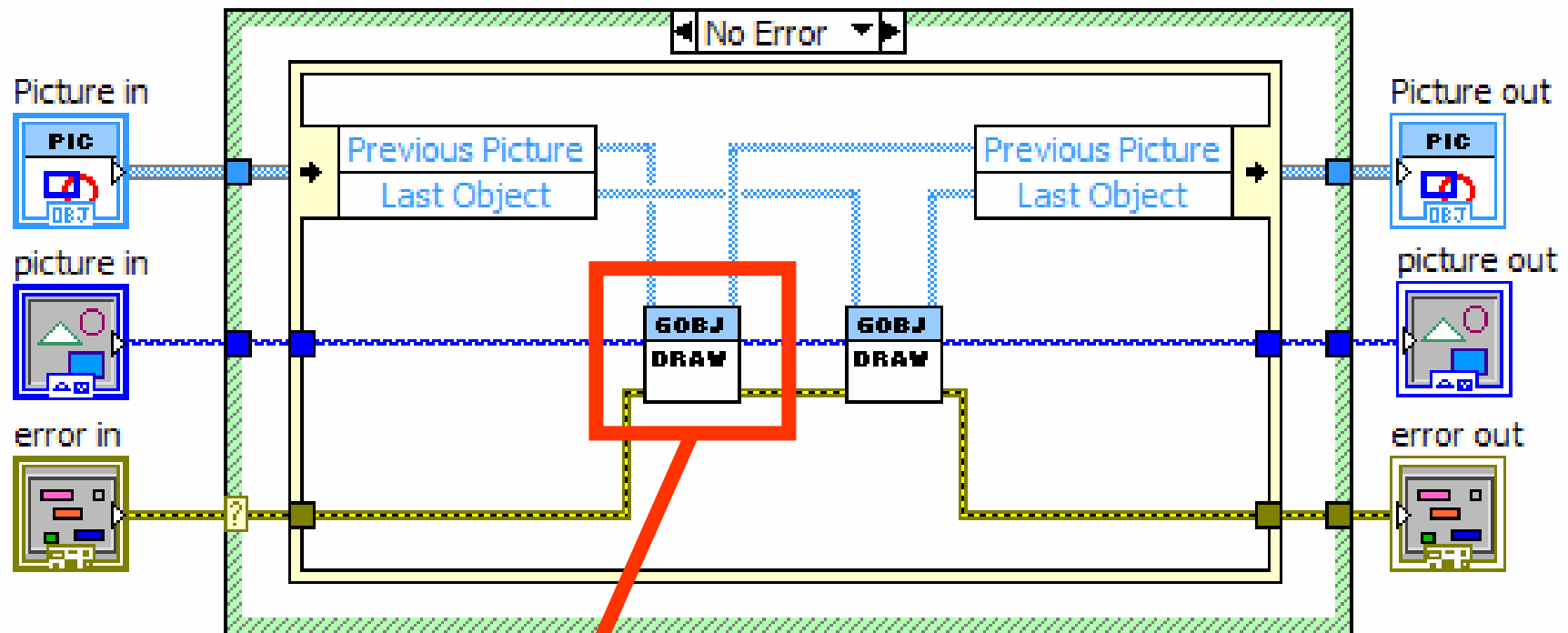
- A dynamic dispatch method VI can call itself directly or indirectly through a call chain

- Requires VI execution priority settings
 - Reentrant execution
 - Share clones between instances



Drawing a Picture Recursively

Picture.lvclass:Draw.vi



Recursive call to this VI itself

Discussion Forum in Finnish Language

- New discussion forum **LabVIEW Finland** opened
<http://expressionflow.com/forums/>
- You're invited to participate any LabVIEW related discussion there in Finnish language

Questions?

Tomi Maila

Editor in Chief

tomi@expressionflow.com

<http://expressionflow.com>

The slides and the example application available online

<http://expressionflow.com/2008/05/07/nidays/>

For presentation related discussions

<http://expressionflow.com/forums/>