Namespace BOOSEUnitTests

Classes

CommandFactoryTests

Unit test for MyCommandFactory to verify it creates command objects based on command types correctly

CommandsTests

Unit tests for individual command classes to verify successful execution and parameters are handled properly

IntegrationTests

Integration tests to verify multiple components work together correctly in complete system workflows

ParserTests

Unit tests for MyParser to verify it correctly pares the program text into executable commands

Class CommandFactoryTests

Namespace: BOOSEUnitTests

Assembly: BOOSEUnitTests.dll

Unit test for MyCommandFactory to verify it creates command objects based on command types correctly

```
[TestClass]
public class CommandFactoryTests
```

Inheritance

object ← CommandFactoryTests

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

MyCommandFactory_CaseInsensitive_CreatesCommands()

Tests that MyCommandFactory handles case-insensitive command types correctly Verifies mixed-case command names still creates the correct command objects

```
[TestMethod]
public void MyCommandFactory_CaseInsensitive_CreatesCommands()
```

MyCommandFactory_CircleCommand_CreatesCorrectCommand()

Tests that MyCommandFactory creates a CircleCommand instance when "circle" command type is requested Verifies factory returns correct command type for circle drawing operations

```
[TestMethod]
public void MyCommandFactory_CircleCommand_CreatesCorrectCommand()
```

MyCommandFactory_DrawToCommand_CreatesCorrectCommand()

Tests that MyCommandFactory creates a DrawToCommand instance when "drawto" command type is requested Verifies factory returns correct command type for line drawing operations

```
[TestMethod]
public void MyCommandFactory_DrawToCommand_CreatesCorrectCommand()
```

MyCommandFactory_EmptyCommand_ReturnsNull()

Tests that MyCommandFactory returns null when an empty string command type is requested Verifies proper handling of empty input strings

```
[TestMethod]
public void MyCommandFactory_EmptyCommand_ReturnsNull()
```

MyCommandFactory_MoveToCommand_CreatesCorrectCommand()

Tests that MyCommandFactory creates a MoveToCommand instance when "moveto" command type is requested Verifies factory returns correct command type for movement operations

```
[TestMethod]
public void MyCommandFactory_MoveToCommand_CreatesCorrectCommand()
```

MyCommandFactory_NullCommand_ReturnsNull()

Tests that MyCommandFactory returns null when a null command type is requested Verifies proper handling for all null values

```
[TestMethod]
public void MyCommandFactory_NullCommand_ReturnsNull()
```

MyCommandFactory_PenCommand_CreatesCorrectCommand()

Tests that MyCommandFactory creates a PenColourCommand instance when "pen" command type is requested Verifies factory returns correct command type for pen colour changing operations

```
[TestMethod]
public void MyCommandFactory_PenCommand_CreatesCorrectCommand()
```

MyCommandFactory_RectCommand_CreatesCorrectCommand()

Tests that MyCommandFactory creates a RectCommand instance when "rect" command type is requested Verifies factory returns correct command type for rectangle drawing operations

```
[TestMethod]
public void MyCommandFactory_RectCommand_CreatesCorrectCommand()
```

MyCommandFactory_UnknownCommand_ReturnsNull()

Tests that MyCommandFactory returns null when an unknown command type is requested Verifies proper handling of invalid command names that do not match any known commands

```
[TestMethod]
public void MyCommandFactory_UnknownCommand_ReturnsNull()
```

MyCommandFactory_WhitespaceCommands_CreatesCommands()

Tests that MyCommandFactory handles command types with leading or trailing whitespace correctly Verifies whitespace trimming funtionality in command parsing

```
[TestMethod]
public void MyCommandFactory_WhitespaceCommands_CreatesCommands()
```

Class CommandsTests

Namespace: BOOSEUnitTests

Assembly: BOOSEUnitTests.dll

Unit tests for individual command classes to verify successful execution and parameters are handled properly

```
[TestClass]
public class CommandsTests
```

Inheritance

object ← CommandsTests

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

CircleCommand_InvalidRadius_ThrowsException()

Tests the CircleCommand throws CommandException when invalid radius is provided Verifies proper error handling for non-numeric radius values

```
[TestMethod]
public void CircleCommand_InvalidRadius_ThrowsException()
```

CircleCommand_NegativeRadius_ThrowsException()

Tests the CircleCommand throws CommandException when negative radius is provided Verifies proper error handling for negative radius values

```
[TestMethod]
public void CircleCommand_NegativeRadius_ThrowsException()
```

CircleCommand_TooManyParameters_ThrowsException()

Tests the CircleCommand throws CommandException when too many parameters are given Verifies the command accepts only one parameter for radius

```
[TestMethod]
public void CircleCommand_TooManyParameters_ThrowsException()
```

CircleCommand_ValidRadius_ExecuteSuccessfully()

Tests the CircleCommand executes successfully when a valid positive radius is provided Verifies circles are drawn with proper radius values

```
[TestMethod]
public void CircleCommand_ValidRadius_ExecuteSuccessfully()
```

CircleCommand_ZeroRadius_ThrowsException()

Tests the CircleCommand throws CommandException when zero radius is provided Verifies that circles must have positive non-Zero radius value

```
[TestMethod]
public void CircleCommand_ZeroRadius_ThrowsException()
```

DrawToCommand_InvaludCoordinates_ThrowsException()

Tests that DrawToCommand throws CommandException when invalid coordinates are provided Verifies proper error handlindd for non-numeric coordinate values

```
[TestMethod]
public void DrawToCommand_InvaludCoordinates_ThrowsException()
```

DrawToCommand_MissingCoordinate()

Tests that DrawToCommand throws CommandException when missing value is provided Verifies the command requires exactly two coordinates

```
[TestMethod]
public void DrawToCommand_MissingCoordinate()
```

DrawToCommand_NegativeCoordinates()

Tests that DrawToCommand throws CommandException when negative coordinates are provided Verifies the commmand validates against negative values

```
[TestMethod]
public void DrawToCommand_NegativeCoordinates()
```

DrawToCommand_ValidCoordinates_Executes()

Tests that DrawToCommand executes successfully when valid parameters are provided Verifies lines can be drawn from current pen position to target coordinates

```
[TestMethod]
public void DrawToCommand_ValidCoordinates_Executes()
```

MoveToCommand_InvalidCoordinates_ThrowsException()

Tests MoveToCommand when invalid coordinates are given a CommandException is thrown Verifies proper error handling when invalid non-numeric parameters are given

```
[TestMethod]
public void MoveToCommand_InvalidCoordinates_ThrowsException()
```

MoveToCommand_MissingParameter_ThrowsException()

Tests the MoveToCommand throws a CommandException when missing parameter is given Verifies the command validates exactly two parameters (x and y)

```
[TestMethod]
public void MoveToCommand_MissingParameter_ThrowsException()
```

MoveToCommand_NegativeCoordinates_ThrowsException()

Tests MoveToCommand throws a CommandException when negative coordinates are provided Verifies the command validates against negative position valuess

```
[TestMethod]
public void MoveToCommand_NegativeCoordinates_ThrowsException()
```

MoveToCommand_TooManyParameters_ThrowsException()

Tests the MoveToCommand throws a CommandException when too many parameters are given Verifies the command validates exactly two parameters (x and y)

```
[TestMethod]
public void MoveToCommand_TooManyParameters_ThrowsException()
```

MoveToCommand_ValidCoordinates_UpdatesPosition()

Tests the MoveToCommand correctly and updates the canvas when valid coordinates are provided Verifies the pen moves to the specified coordinates (x and y) position without drawing

```
[TestMethod]
public void MoveToCommand_ValidCoordinates_UpdatesPosition()
```

PenColourCommand_InvalidColours_ThrowsException()

Tests the PenColourCommand throws CommandException when invalid or out of range values are provided Verifies RGB values must be positive and between 0-255

```
[TestMethod]
public void PenColourCommand_InvalidColours_ThrowsException()
```

PenColourCommand_MissingParameter_ThrowsException()

Tests the PenColourCommand throws CommandException when insufficient colour parameters are provided Verifies the command requires exactly three RGB values

```
[TestMethod]
public void PenColourCommand_MissingParameter_ThrowsException()
```

PenColourCommand_MissingTwoParameters_ThrowsException()

Tests the PenColourCommand throws CommandException when 2 parameters are missing Verifies the command requires all three RGB values

```
[TestMethod]
public void PenColourCommand_MissingTwoParameters_ThrowsException()
```

PenColourCommand_OutOfRangeColours_ThrowsException()

Tests the PenColourCommand throws CommandException when multiple out of range values are provided Verifies comprehensive RGB value Validation

```
[TestMethod]
public void PenColourCommand_OutOfRangeColours_ThrowsException()
```

PenColourCommand_TooManyParameters_ThrowsException()

Tests the PenColourCommand throws CommandException when more than three parameters are provided Verifies the command accepts exactly three parameters

```
[TestMethod]
public void PenColourCommand_TooManyParameters_ThrowsException()
```

PenColourCommand_ValidColours_Executes()

Tests the PenColourCommand executes successfully when valid RGB values are given Verifies pen colour can be changed with correct colour values

```
[TestMethod]
public void PenColourCommand_ValidColours_Executes()
```

RectCommand_InvalidDimensions_ThrowsException()

Tests that RectCommand throws CommandException when invalid dimensions are provided Verifies proper error handling for negative or invalid dimensions

```
[TestMethod]
public void RectCommand_InvalidDimensions_ThrowsException()
```

RectCommand_InvalidInput_ThrowsException()

Tests that RectCommand throws CommandException when a non-numeric input is provided Verifies propeer error handling for non-numeric text based invalid parameters

```
[TestMethod]
public void RectCommand_InvalidInput_ThrowsException()
```

RectCommand_MissingParameter_ThrowsException()

Tests that RectCommand throws CommandException when insufficient parameters are provided Verifies the command requires exactly two parameters

```
[TestMethod]
public void RectCommand_MissingParameter_ThrowsException()
```

RectCommand_TooManyParameters_ThrowsException()

Tests that RectCommand throws CommandException when too many parameters are provided Verifies the command only accepts 2 dimensions (Height and Width)

```
[TestMethod]
public void RectCommand_TooManyParameters_ThrowsException()
```

RectCommand_ValidDimensions_ExecuteSuccessfully()

Tests the RectCommand executes successfully when valid dimensions are provided Verifies rectangles can be drawn with correct height and width values

```
[TestMethod]
public void RectCommand_ValidDimensions_ExecuteSuccessfully()
```

RectCommand_ZeroDimensions_ThrowsException()

Tests that RectCommand throws CommandException when zero dimensions are provided Verifies rectangles dimensions must be positive non-Zero values

```
[TestMethod]
public void RectCommand_ZeroDimensions_ThrowsException()
```

Class IntegrationTests

Namespace: BOOSEUnitTests

Assembly: BOOSEUnitTests.dll

Integration tests to verify multiple components work together correctly in complete system workflows

```
[TestClass]
public class IntegrationTests
```

Inheritance

object ← IntegrationTests

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AssignmentProgram_1UnrestrictedBoose_ExecutesSuccessfully()

Tests the exact 1UnrestrictedBoose program from assignment requirements executes successfully Verifies the complete system can handle the assignment's core test case

```
[TestMethod]
public void AssignmentProgram_1UnrestrictedBoose_ExecutesSuccessfully()
```

AssignmentProgram_NoRestrictionsProven_NoArtficialLimits()

Tests that the intepreter has no artificial restrictions by using large dimensions and coordinates Verifies the custom interpreter removes limitations present in the original BOOSE system

```
[TestMethod]
public void AssignmentProgram_NoRestrictionsProven_NoArtficialLimits()
```

ColourChangingPattern_DifferentColours_WorksSuccessfully()

Tests complex colour changing patterns accross multiple drawing operations Verifies that system handles frequent pen colour changed correctly

```
[TestMethod]
public void ColourChangingPattern_DifferentColours_WorksSuccessfully()
```

ComplexProgram_WithComments_IgnoresCommentsSuccessfully()

Tests that the parser correctly ignores various comment styles during program execution Verifies multiple comment types (//,*,**,--) are ignored while commands execute

```
[TestMethod]
public void ComplexProgram_WithComments_IgnoresCommentsSuccessfully()
```

LargeProgram_TooManyCommands_ExecutesWithoutMemoryIssue()

Tests system perfromance and memory handling with a large number of commands Verifies that the intepereter can handle 100+ commands without any memory issues or crashes

```
[TestMethod]
public void LargeProgram_TooManyCommands_ExecutesWithoutMemoryIssue()
```

MixedShapes_Variety_DrawsSuccessfully()

Tests a variety of ramdom shapes and operations in a single program Verifies the system handles mixed command types seamlessly

```
[TestMethod]
public void MixedShapes_Variety_DrawsSuccessfully()
```

MultilineProgram_ValidCommands_ExecutesSuccessfully()

Tests that a multiline program with multiple commands executes successfully end-to-end Verifies complete workflow from parsing to execution for standard drawng sequences

```
[TestMethod]
public void MultilineProgram_ValidCommands_ExecutesSuccessfully()
```

ProgramExecution_ErrorBetweenCommands_ReportsCorrectLine()

Tests that the parser correctly indentifies errors between the commands with correct line number Verifies error handling when invalid commands are mixed with valid commands in the program

```
[TestMethod]
public void ProgramExecution_ErrorBetweenCommands_ReportsCorrectLine()
```

RandomShapes_MixedColours_DrawsSuccessfully()

Tests that random shapes with a mix of colours can be drawn successfully in sequence Verifies that system handles diverse draing operations without any issues

```
[TestMethod]
public void RandomShapes_MixedColours_DrawsSuccessfully()
```

SimpleDrawing_BasicLines_DrawsSuccessfully()

Tests basic line drawing functionality using multiple DrawTo commands Verifies connected line sequences render correctly

```
[TestMethod]
public void SimpleDrawing_BasicLines_DrawsSuccessfully()
```

Class ParserTests

Namespace: BOOSEUnitTests

Assembly: BOOSEUnitTests.dll

Unit tests for MyParser to verify it correctly pares the program text into executable commands

```
[TestClass]
public class ParserTests
```

Inheritance

object ← ParserTests

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

MultiLineProgram_ValidCommands_Executes()

Tests that MyParser correctly parses a multiline program with valid commands Verifies all commands are properly extracted and added to the proogram

```
[TestMethod]
public void MultiLineProgram_ValidCommands_Executes()
```

Parser_EmptyProgram_DoesNothing()

Tests that MyParser handles empty program text gracefully Verifies no commands are created and no errors occur with empty input

```
[TestMethod]
public void Parser_EmptyProgram_DoesNothing()
```

Parser_ExtraWhiteSpace_ParseSucessfully()

Tests MyParser correctly handles programs with extra whitespaces Verifies good parsing with irregular spacing between commands and parameters

```
[TestMethod]
public void Parser_ExtraWhiteSpace_ParseSucessfully()
```

Parser_InvalidCommand_ThrowsException()

Tests that MyParser throws CommandException when encoutring invalid commands Verifies proper error handling for unknown command types in the program text

```
[TestMethod]
public void Parser_InvalidCommand_ThrowsException()
```

Parser_MixedCaseCommands_ParseSuccessfully()

Tests that MyParser handles programs with mixed case commands correctly Veriifies case-insensitive command recognition during parsing

```
[TestMethod]
public void Parser_MixedCaseCommands_ParseSuccessfully()
```

Parser_ProgramWithComments_IgnoresComments()

Tests that MyParser correctly ingores comment lines during parsing Verifies different comment styles (//,--,*) are properly skipped

```
[TestMethod]
public void Parser_ProgramWithComments_IgnoresComments()
```

Namespace BOOSEapp

Classes

AppCanvas

Implements the BOOSE ICanvas interface for drawing operations Handles all graphic operations like drawing shapes, lines, and managing colours This is the actual canvas that the BOOSE system uses for rendering

CommandException

Custom exceprion for command related errors in the custom BOOSE interpreter Used when commands have problems with parameters or execution

Form1

Main application window for the BOOSE drawing interpreter Handles UI, user input, and coordinates between parser and canvas

MoveToCommand

Handles the moveto command, it moves the cursor wuthout drawing anything Usage: moveto 200,100

MyCommandFactory

Creates command objects based on command names This is the "command creator" is knows how to make each type of command

MyParser

MyStoredProgram

Custom program storage that holds and executes drawing commands This is where all the commands live before they get executed

Interfaces

ICommandHandler

Interface all command classes must implement Defines the basic structre for every drawing command

Class AppCanvas

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Implements the BOOSE ICanvas interface for drawing operations Handles all graphic operations like drawing shapes, lines, and managing colours This is the actual canvas that the BOOSE system uses for rendering

```
public class AppCanvas : ICanvas
```

Inheritance

object ← AppCanvas

Implements

ICanvas

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AppCanvas(int, int)

Creates a new drawing canvas with specified dimensions Sets up the bitmap, graphics object, and default drawing settings

```
public AppCanvas(int xsize, int ysize)
```

Parameters

`xsize` int

Width of the canvas in pixels

`ysize` int

Height of the canvas in pixels

Properties

PenColour

Gets or sets the current colour for drawing operations

```
public object PenColour { get; set; }
```

Property Value

object

Xpos

Gets or sets the current X position of the drawing cursor

```
public int Xpos { get; set; }
```

Property Value

int

Ypos

Gets or sets the current Y position of the drawing cursor

```
public int Ypos { get; set; }
```

Property Value

int

Methods

Circle(int, bool)

Draws a circle (outline or filled) centered at current position

```
public void Circle(int radius, bool filled)
```

Parameters

`radius` int

Size of the circle - distance from center to edge

`filled` bool

True for solid circle, false for outline only

Clear()

Clears the entire canvas to white - wipes everything clean

```
public void Clear()
```

DrawTo(int, int)

Draws a straight line from current position to specified coordinates Uses the current pen colour and thickness, then updates position to end point

```
public void DrawTo(int x, int y)
```

Parameters

`x` int

Target X coordinate for line end

`y` int

Target Y coordinate for line end

MoveTo(int, int)

Moves the drawing cursor to new coordinates without drawing anything It just changes the position for the next drawing operation

```
public void MoveTo(int x, int y)
```

Parameters

`x` int

New X position

`y` int

New Y position

Rect(int, int, bool)

Draws a rectangle (Outline or Filled) starting from the current position Current position becomes the top-left corner of the rectangle

```
public void Rect(int width, int height, bool filled)
```

Parameters

`width` int

How wide the rectangle should be

`height` int

How high the rectangle should be

`filled` bool

True for solid rectangle, false for outline only

Reset()

Resets the drawing cursor back to its origin (0,0) Does not clear the canvas, only returns the cursor to its oirigin

```
public void Reset()
```

Set(int, int)

Resizes the canvas to new dimensions and clears it Useful if the window size changes

```
public void Set(int width, int height)
```

Parameters

`width` int

New canvas width

`height` int

New canvas height

SetColour(int, int, int)

Changes the drawing colour using RGB values Affects all future drawing operations until changed

```
public void SetColour(int red, int green, int blue)
```

Parameters

`red` int

Red component (0-255)

`green` int

Green component (0-255)

`blue` int

Blue component (0-255)

Tri(int, int)

Draws a triangle centered at current position Creates an equilateral triangle with specified width and height

```
public void Tri(int width, int height)
```

Parameters

`width` int

Base width of the triangle

`height` int

Height of the triangle from the base to the tip

WriteText(string)

Draws text on the canvas at current position Uses current pen colour for the text colour

```
public void WriteText(string text)
```

Parameters

`text` string

The text string to draw

getBitmap()

Gets the current bitmap image of the canvas - what is actually drawn

```
public object getBitmap()
```

Returns

object

Bitmap containing all drawn shapes

Class CommandException

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Custom exceprion for command related errors in the custom BOOSE interpreter Used when commands have problems with parameters or execution

```
public class CommandException : Exception, ISerializable
```

Inheritance

object ← Exception ← CommandException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CommandException(string)

Creates a new command exception with an error message

```
public CommandException(string message)
```

Parameters

`message` string

Description of what went wrong during execution

CommandException(string, Exception)

Creates a new command exception with message and inner exception Useful when wrapping another error that occured

```
public CommandException(string message, Exception inner)
```

Parameters

`message` string

Description of what went wrong during execution

`inner` Exception

The original exception that caused this error

Class Form1

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Main application window for the BOOSE drawing interpreter Handles UI, user input, and coordinates between parser and canvas

```
public class Form1 : Form, IDropTarget, ISynchronizeInvoke, IWin32Window,
IBindableComponent, IComponent, IDisposable, IContainerControl
```

Inheritance

object ← MarshalByRefObject ← Component ← Control ← ScrollableControl ← ContainerControl ← Form ← Form1

Implements

IDropTarget, ISynchronizeInvoke, IWin32Window, IBindableComponent, IComponent, IDisposable, IContainerControl

Inherited Members

Form.SetVisibleCore(bool) , Form.Activate() , Form.ActivateMdiChild(Form) , Form.AddOwnedForm(Form) , Form.AdjustFormScrollbars(bool) , Form.Close() , Form.CreateAccessibilityInstance() , Form.CreateControlsInstance() , Form.CreateHandle() , Form.DefWndProc(ref Message) , Form.ProcessMnemonic(char) , Form.CenterToParent() , Form.CenterToScreen() , Form.LayoutMdi(MdiLayout) , Form.OnActivated(EventArgs) , Form.OnBackgroundImageChanged(EventArgs) , Form.OnBackgroundImageLayoutChanged(EventArgs) , Form.OnClosing(CancelEventArgs) , Form.OnClosed(EventArgs) , Form.OnFormClosing(FormClosingEventArgs) , Form.OnFormClosed(FormClosedEventArgs) , Form.OnCreateControl() , Form.OnDeactivate(EventArgs) , Form.OnEnabledChanged(EventArgs) , Form.OnEnter(EventArgs) , Form.OnFontChanged(EventArgs) , Form.OnGotFocus(EventArgs) , Form.OnHandleCreated(EventArgs) , Form.OnHandleDestroyed(EventArgs) , Form.OnHelpButtonClicked(CancelEventArgs) , Form.OnLayout(LayoutEventArgs) , Form.OnLoad(EventArgs) , Form.OnMaximizedBoundsChanged(EventArgs) , Form.OnMaximumSizeChanged(EventArgs) , Form.OnMinimumSizeChanged(EventArgs) , Form.OnInputLanguageChanged(InputLanguageChangedEventArgs) , Form.OnInputLanguageChanging(InputLanguageChangingEventArgs) , Form.OnVisibleChanged(EventArgs) , Form.OnMdiChildActivate(EventArgs) , Form.OnMenuStart(EventArgs) , Form.OnMenuComplete(EventArgs) , Form.OnPaint(PaintEventArgs) , Form.OnResize(EventArgs) ,

Form.OnDpiChanged(DpiChangedEventArgs) , Form.OnGetDpiScaledSize(int, int, ref Size) ,
Form.OnRightToLeftLayoutChanged(EventArgs) , Form.OnShown(EventArgs) ,
Form.OnTextChanged(EventArgs) , Form.ProcessCmdKey(ref Message, Keys) ,
Form.ProcessDialogKey(Keys) , Form.ProcessDialogChar(char) ,
Form.ProcessKeyPreview(ref Message) , Form.ProcessTabKey(bool) ,
Form.RemoveOwnedForm(Form) , Form.Select(bool, bool) ,
Form.ScaleMinMaxSize(float, float, bool) ,
Form.GetScaledBounds(Rectangle, SizeF, BoundsSpecified) ,
Form.ScaleControl(SizeF, BoundsSpecified) , Form.SetBoundsCore(int, int, int, int, BoundsSpecified) ,
Form.SetClientSizeCore(int, int) , Form.SetDesktopBounds(int, int, int, int) ,
Form.SetDesktopLocation(int, int) , Form.Show(IWin32Window) , Form.ShowDialog() ,
Form.ShowDialog(IWin32Window) , Form.ToString() , Form.UpdateDefaultButton() ,
Form.OnResizeBegin(EventArgs) , Form.OnResizeEnd(EventArgs) ,
Form.OnStyleChanged(EventArgs) , Form.ValidateChildren() ,
Form.ValidateChildren(ValidationConstraints) , Form.WndProc(ref Message) , Form.AcceptButton ,
Form.ActiveForm , Form.ActiveMdiChild , Form.AllowTransparency , Form.AutoScroll ,
Form.AutoSize , Form.AutoSizeMode , Form.AutoValidate , Form.BackColor ,
Form.FormBorderStyle , Form.CancelButton , Form.ClientSize , Form.ControlBox ,
Form.CreateParams , Form.DefaultImeMode , Form.DefaultSize , Form.DesktopBounds ,
Form.DesktopLocation , Form.DialogResult , Form.HelpButton , Form.Icon , Form.IsMdiChild ,
Form.IsMdiContainer , Form.IsRestrictedWindow , Form.KeyPreview , Form.Location ,
Form.MaximizedBounds , Form.MaximumSize , Form.MainMenuStrip , Form.MinimumSize ,
Form.MaximizeBox , Form.MdiChildren , Form.MdiChildrenMinimizedAnchorBottom ,
Form.MdiParent , Form.MinimizeBox , Form.Modal , Form.Opacity , Form.OwnedForms ,
Form.Owner , Form.RestoreBounds , Form.RightToLeftLayout , Form.ShowInTaskbar ,
Form.ShowIcon , Form.ShowWithoutActivation , Form.Size , Form.SizeGripStyle ,
Form.StartPosition , Form.Text , Form.TopLevel , Form.TopMost , Form.TransparencyKey ,
Form.WindowState , Form.AutoSizeChanged , Form.AutoValidateChanged ,
Form.HelpButtonClicked , Form.MaximizedBoundsChanged , Form.MaximumSizeChanged ,
Form.MinimumSizeChanged , Form.Activated , Form.Deactivate , Form.FormClosing ,
Form.FormClosed , Form.Load , Form.MdiChildActivate , Form.MenuComplete ,
Form.MenuStart , Form.InputLanguageChanged , Form.InputLanguageChanging ,
Form.RightToLeftLayoutChanged , Form.Shown , Form.DpiChanged , Form.ResizeBegin ,
Form.ResizeEnd , ContainerControl.OnAutoValidateChanged(EventArgs) ,
ContainerControl.OnMove(EventArgs) , ContainerControl.OnParentChanged(EventArgs) ,
ContainerControl.PerformAutoScale() , ContainerControl.RescaleConstantsForDpi(int, int) ,
ContainerControl.Validate() , ContainerControl.Validate(bool) ,
ContainerControl.AutoScaleDimensions , ContainerControl.AutoScaleFactor ,
ContainerControl.AutoScaleMode , ContainerControl.BindingContext ,
ContainerControl.CanEnableIme , ContainerControl.ActiveControl ,

ContainerControl.CurrentAutoScaleDimensions , ContainerControl.ParentForm ,
ScrollableControl.ScrollStateAutoScrolling , ScrollableControl.ScrollStateHScrollVisible ,
ScrollableControl.ScrollStateVScrollVisible , ScrollableControl.ScrollStateUserHasScrolled ,
ScrollableControl.ScrollStateFullDrag , ScrollableControl.GetScrollState(int) ,
ScrollableControl.OnMouseWheel(MouseEventArgs) ,
ScrollableControl.OnRightToLeftChanged(EventArgs) ,
ScrollableControl.OnPaintBackground(PaintEventArgs) ,
ScrollableControl.OnPaddingChanged(EventArgs) , ScrollableControl.SetDisplayRectLocation(int, int) ,
ScrollableControl.ScrollControlIntoView(Control) , ScrollableControl.ScrollToControl(Control) ,
ScrollableControl.OnScroll(ScrollEventArgs) , ScrollableControl.SetAutoScrollMargin(int, int) ,
ScrollableControl.SetScrollState(int, bool) , ScrollableControl.AutoScrollMargin ,
ScrollableControl.AutoScrollPosition , ScrollableControl.AutoScrollMinSize ,
ScrollableControl.DisplayRectangle , ScrollableControl.HScroll , ScrollableControl.HorizontalScroll ,
ScrollableControl.VScroll , ScrollableControl.VerticalScroll , ScrollableControl.Scroll ,
Control.GetAccessibilityObjectById(int) , Control.SetAutoSizeMode(AutoSizeMode) ,
Control.GetAutoSizeMode() , Control.GetPreferredSize(Size) ,
Control.AccessibilityNotifyClients(AccessibleEvents, int) ,
Control.AccessibilityNotifyClients(AccessibleEvents, int, int) , Control.BeginInvoke(Delegate) ,
Control.BeginInvoke(Action) , Control.BeginInvoke(Delegate, params object[]) ,
Control.BringToFront() , Control.Contains(Control) , Control.CreateGraphics() ,
Control.CreateControl() , Control.DestroyHandle() , Control.DoDragDrop(object, DragDropEffects) ,
Control.DoDragDrop(object, DragDropEffects, Bitmap, Point, bool) ,
Control.DrawToBitmap(Bitmap, Rectangle) , Control.EndInvoke(IAsyncResult) , Control.FindForm() ,
Control.GetTopLevel() , Control.RaiseKeyEvent(object, KeyEventArgs) ,
Control.RaiseMouseEvent(object, MouseEventArgs) , Control.Focus() ,
Control.FromChildHandle(nint) , Control.FromHandle(nint) ,
Control.GetChildAtPoint(Point, GetChildAtPointSkip) , Control.GetChildAtPoint(Point) ,
Control.GetContainerControl() , Control.GetNextControl(Control, bool) ,
Control.GetStyle(ControlStyles) , Control.Hide() , Control.InitLayout() , Control.Invalidate(Region) ,
Control.Invalidate(Region, bool) , Control.Invalidate() , Control.Invalidate(bool) ,
Control.Invalidate(Rectangle) , Control.Invalidate(Rectangle, bool) , Control.Invoke(Action) ,
Control.Invoke(Delegate) , Control.Invoke(Delegate, params object[]) ,
Control.Invoke<T>(Func<T>) , Control.InvokePaint(Control, PaintEventArgs) ,
Control.InvokePaintBackground(Control, PaintEventArgs) , Control.IsKeyLocked(Keys) ,
Control.IsInputChar(char) , Control.IsInputKey(Keys) , Control.IsMnemonic(char, string) ,
Control.LogicalToDeviceUnits(int) , Control.LogicalToDeviceUnits(Size) ,
Control.ScaleBitmapLogicalToDevice(ref Bitmap) , Control.NotifyInvalidate(Rectangle) ,
Control.InvokeOnClick(Control, EventArgs) , Control.OnAutoSizeChanged(EventArgs) ,
Control.OnBackColorChanged(EventArgs) , Control.OnBindingContextChanged(EventArgs) ,
Control.OnCausesValidationChanged(EventArgs) , Control.OnContextMenuStripChanged(EventArgs) ,

Control.OnCursorChanged(EventArgs) , Control.OnDataContextChanged(EventArgs) ,
Control.OnDockChanged(EventArgs) , Control.OnForeColorChanged(EventArgs) ,
Control.OnNotifyMessage(Message) , Control.OnParentBackColorChanged(EventArgs) ,
Control.OnParentBackgroundImageChanged(EventArgs) ,
Control.OnParentBindingContextChanged(EventArgs) , Control.OnParentCursorChanged(EventArgs) ,
Control.OnParentDataContextChanged(EventArgs) , Control.OnParentEnabledChanged(EventArgs) ,
Control.OnParentFontChanged(EventArgs) , Control.OnParentForeColorChanged(EventArgs) ,
Control.OnParentRightToLeftChanged(EventArgs) , Control.OnParentVisibleChanged(EventArgs) ,
Control.OnPrint(PaintEventArgs) , Control.OnTabIndexChanged(EventArgs) ,
Control.OnTabStopChanged(EventArgs) , Control.OnClick(EventArgs) ,
Control.OnClientSizeChanged(EventArgs) , Control.OnControlAdded(ControlEventArgs) ,
Control.OnControlRemoved(ControlEventArgs) , Control.OnLocationChanged(EventArgs) ,
Control.OnDoubleClick(EventArgs) , Control.OnDragEnter(DragEventArgs) ,
Control.OnDragOver(DragEventArgs) , Control.OnDragLeave(EventArgs) ,
Control.OnDragDrop(DragEventArgs) , Control.OnGiveFeedback(GiveFeedbackEventArgs) ,
Control.InvokeGotFocus(Control, EventArgs) , Control.OnHelpRequested(HelpEventArgs) ,
Control.OnInvalidated(InvalidateEventArgs) , Control.OnKeyDown(KeyEventArgs) ,
Control.OnKeyPress(KeyPressEventArgs) , Control.OnKeyUp(KeyEventArgs) ,
Control.OnLeave(EventArgs) , Control.InvokeLostFocus(Control, EventArgs) ,
Control.OnLostFocus(EventArgs) , Control.OnMarginChanged(EventArgs) ,
Control.OnMouseDoubleClick(MouseEventArgs) , Control.OnMouseClick(MouseEventArgs) ,
Control.OnMouseCaptureChanged(EventArgs) , Control.OnMouseDown(MouseEventArgs) ,
Control.OnMouseEnter(EventArgs) , Control.OnMouseLeave(EventArgs) ,
Control.OnDpiChangedBeforeParent(EventArgs) , Control.OnDpiChangedAfterParent(EventArgs) ,
Control.OnMouseHover(EventArgs) , Control.OnMouseMove(MouseEventArgs) ,
Control.OnMouseUp(MouseEventArgs) ,
Control.OnQueryContinueDrag(QueryContinueDragEventArgs) ,
Control.OnRegionChanged(EventArgs) , Control.OnPreviewKeyDown(PreviewKeyDownEventArgs) ,
Control.OnSizeChanged(EventArgs) , Control.OnChangeUICues(UICuesEventArgs) ,
Control.OnSystemColorsChanged(EventArgs) , Control.OnValidating(CancelEventArgs) ,
Control.OnValidated(EventArgs) , Control.PerformLayout() , Control.PerformLayout(Control, string) ,
Control.PointToClient(Point) , Control.PointToScreen(Point) ,
Control.PreProcessMessage(ref Message) , Control.PreProcessControlMessage(ref Message) ,
Control.ProcessKeyEventArgs(ref Message) , Control.ProcessKeyMessage(ref Message) ,
Control.RaiseDragEvent(object, DragEventArgs) , Control.RaisePaintEvent(object, PaintEventArgs) ,
Control.RecreateHandle() , Control.RectangleToClient(Rectangle) ,
Control.RectangleToScreen(Rectangle) , Control.ReflectMessage(nint, ref Message) ,
Control.Refresh() , Control.ResetMouseEventArgs() , Control.ResetText() , Control.ResumeLayout() ,
Control.ResumeLayout(bool) , Control.Scale(SizeF) , Control.Select() ,
Control.SelectNextControl(Control, bool, bool, bool, bool) , Control.SendToBack() ,

Control.SetBounds(int, int, int, int) , Control.SetBounds(int, int, int, int, BoundsSpecified) ,
Control.SizeFromClientSize(Size) , Control.SetStyle(ControlStyles, bool) , Control.SetTopLevel(bool) ,
Control.RtlTranslateAlignment(HorizontalAlignment) ,
Control.RtlTranslateAlignment(LeftRightAlignment) ,
Control.RtlTranslateAlignment(ContentAlignment) ,
Control.RtlTranslateHorizontal(HorizontalAlignment) ,
Control.RtlTranslateLeftRight(LeftRightAlignment) , Control.RtlTranslateContent(ContentAlignment) ,
Control.Show() , Control.SuspendLayout() , Control.Update() , Control.UpdateBounds() ,
Control.UpdateBounds(int, int, int, int) , Control.UpdateBounds(int, int, int, int, int, int) ,
Control.UpdateZOrder() , Control.UpdateStyles() , Control.OnImeModeChanged(EventArgs) ,
Control.AccessibilityObject , Control.AccessibleDefaultActionDescription ,
Control.AccessibleDescription , Control.AccessibleName , Control.AccessibleRole ,
Control.AllowDrop , Control.Anchor , Control.AutoScrollOffset , Control.LayoutEngine ,
Control.DataContext , Control.BackgroundImage , Control.BackgroundImageLayout ,
Control.Bottom , Control.Bounds , Control.CanFocus , Control.CanRaiseEvents ,
Control.CanSelect , Control.Capture , Control.CausesValidation ,
Control.CheckForIllegalCrossThreadCalls , Control.ClientRectangle , Control.CompanyName ,
Control.ContainsFocus , Control.ContextMenuStrip , Control.Controls , Control.Created ,
Control.Cursor , Control.DataBindings , Control.DefaultBackColor , Control.DefaultCursor ,
Control.DefaultFont , Control.DefaultForeColor , Control.DefaultMargin ,
Control.DefaultMaximumSize , Control.DefaultMinimumSize , Control.DefaultPadding ,
Control.DeviceDpi , Control.IsDisposed , Control.Disposing , Control.Dock ,
Control.DoubleBuffered , Control.Enabled , Control.Focused , Control.Font ,
Control.FontHeight , Control.ForeColor , Control.Handle , Control.HasChildren , Control.Height ,
Control.IsHandleCreated , Control.InvokeRequired , Control.IsAccessible ,
Control.IsAncestorSiteInDesignMode , Control.IsMirrored , Control.Left , Control.Margin ,
Control.ModifierKeys , Control.MouseButtons , Control.MousePosition , Control.Name ,
Control.Parent , Control.ProductName , Control.ProductVersion , Control.RecreatingHandle ,
Control.Region , Control.RenderRightToLeft , Control.ResizeRedraw , Control.Right ,
Control.RightToLeft , Control.ScaleChildren , Control.Site , Control.TabIndex , Control.TabStop ,
Control.Tag , Control.Top , Control.TopLevelControl , Control.ShowKeyboardCues ,
Control.ShowFocusCues , Control.UseWaitCursor , Control.Visible , Control.Width ,
Control.PreferredSize , Control.Padding , Control.ImeMode , Control.ImeModeBase ,
Control.PropagatingImeMode , Control.BackColorChanged , Control.BackgroundImageChanged ,
Control.BackgroundImageLayoutChanged , Control.BindingContextChanged ,
Control.CausesValidationChanged , Control.ClientSizeChanged ,
Control.ContextMenuStripChanged , Control.CursorChanged , Control.DockChanged ,
Control.EnabledChanged , Control.FontChanged , Control.ForeColorChanged ,
Control.LocationChanged , Control.MarginChanged , Control.RegionChanged ,
Control.RightToLeftChanged , Control.SizeChanged , Control.TabIndexChanged ,

Control.TabStopChanged , Control.TextChanged , Control.VisibleChanged , Control.Click , Control.ControlAdded , Control.ControlRemoved , Control.DataContextChanged , Control.DragDrop , Control.DragEnter , Control.DragOver , Control.DragLeave , Control.GiveFeedback , Control.HandleCreated , Control.HandleDestroyed , Control.HelpRequested , Control.Invalidated , Control.PaddingChanged , Control.Paint , Control.QueryContinueDrag , Control.QueryAccessibilityHelp , Control.DoubleClick , Control.Enter , Control.GotFocus , Control.KeyDown , Control.KeyPress , Control.KeyUp , Control.Layout , Control.Leave , Control.LostFocus , Control.MouseClick , Control.MouseDoubleClick , Control.MouseCaptureChanged , Control.MouseDown , Control.MouseEnter , Control.MouseLeave , Control.DpiChangedBeforeParent , Control.DpiChangedAfterParent , Control.MouseHover , Control.MouseMove , Control.MouseUp , Control.MouseWheel , Control.Move , Control.PreviewKeyDown , Control.Resize , Control.ChangeUICues , Control.StyleChanged , Control.SystemColorsChanged , Control.Validating , Control.Validated , Control.ParentChanged , Control.ImeModeChanged , Component.Dispose() , Component.GetService(Type) , Component.Container , Component.DesignMode , Component.Events , Component.Disposed , MarshalByRefObject.GetLifetimeService() , MarshalByRefObject.InitializeLifetimeService() , MarshalByRefObject.MemberwiseClone(bool) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Form1()

Intialises the main form and sets up the drawing system Creates a canvas, command factory, program storage and parser

```
public Form1()
```

Methods

Dispose(bool)

Clean up any resources being used.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

true if managed resources should be disposed; otherwise, false.

Interface ICommandHandler

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Interface all command classes must implement Defines the basic structre for every drawing command

```
public interface ICommandHandler
```

Methods

Compile()

Validates and prepares the command for execution Checks of the parameters are correct and converts them to their proper types

```
void Compile()
```

Execute()

Performs the actual drawing operation on the canvas This is where the command executes and does its real work

```
void Execute()
```

Set(StoredProgram, string)

Sets up the command with program reference and parses parameters This is where the command get its data ready for execution

```
void Set(StoredProgram program, string parameters)
```

Parameters

`program` StoredProgram

The main program that owns this command

`parameters` string

String containing command parameters

Class MoveToCommand

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Handles the moveto command, it moves the cursor wuthout drawing anything Usage: moveto 200,100

```
public class MoveToCommand : ICommand, ICommandHandler
```

Inheritance

object ← MoveToCommand

Implements

ICommand, ICommandHandler

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MoveToCommand(ICanvas)

Creates a moveto command with the drawing canvas

```
public MoveToCommand(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas to draw on

Methods

CheckParameters(string[])

Checks if the parameters are valid it needs exactly 2 parameters (x and y)

```
public void CheckParameters(string[] parameters)
```

Parameters

`parameters` string[]

Should have 2 values: [x and y]

Exceptions

CommandException

Thrown when a wrong number of parameters is given

Compile()

Parses and validates the coordinates it also make sure that they are positive numbers

```
public void Compile()
```

Exceptions

CommandException

Thrown if the coordinates are invalid or negative numbers

Execute()

Moves the cursor to the specified coordinates No visible line is drawn - Just changes the position

```
public void Execute()
```

Set(StoredProgram, string)

Sets up the command with the program and parses the parameters

```
public void Set(StoredProgram Program, string parameterList)
```

Parameters

`Program` StoredProgram

The program that this command belongs to

`parameterList` string

String with comma separated parameters like 100,200

Class MyCommandFactory

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Creates command objects based on command names This is the "command creator" is knows how to make each type of command

```
public class MyCommandFactory
```

Inheritance

object ← MyCommandFactory

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MyCommandFactory(ICanvas)

Creates a new factory linked to a drawing canvas

```
public MyCommandFactory(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas that commands will draw on

Methods

MakeCommand(string)

Creates the correct command object for a given command type Looks at the command name and returns the appropriate command class

```
public ICommandHandler MakeCommand(string commandType)
```

Parameters

`commandType` string

The command name (moveto, drawto, circle, rect, etc...)

Returns

ICommandHandler

Class MyParser

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

```
public class MyParser
```

Inheritance

object ← MyParser

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MyParser(MyCommandFactory, MyStoredProgram)

Creates a new parser with command factory and program storage

```
public MyParser(MyCommandFactory factory, MyStoredProgram program)
```

Parameters

`factory` MyCommandFactory

Used to create command objects

`program` MyStoredProgram

Where the parsed commands will be stored

Methods

ParseProgram(string)

Reads program text and converts it into command objects Handles multiple lines, comments, and error checking

```
public void ParseProgram(string programText)
```

Parameters

`programText` string

The raw text of the BOOSE program

Exceptions

CommandException

Thrown if parsing fails

Class MyStoredProgram

Namespace: BOOSEapp

Assembly: BOOSEapp.dll

Custom program storage that holds and executes drawing commands This is where all the commands live before they get executed

```
public class MyStoredProgram : StoredProgram, IList, ICollection, IEnumerable,
ICloneable, IStoredProgram
```

Inheritance

object ← ArrayList ← StoredProgram ← MyStoredProgram

Implements

IList, ICollection, IEnumerable, ICloneable, IStoredProgram

Inherited Members

StoredProgram.IsValidProgram() , StoredProgram.SetSyntaxStatus(bool) , StoredProgram.AddMethod(Method) , StoredProgram.GetMethod(string) , StoredProgram.AddVariable(Evaluation) , StoredProgram.GetVariable(string) , StoredProgram.GetVariable(int) , StoredProgram.FindVariable(Evaluation) , StoredProgram.FindVariable(string) , StoredProgram.VariableExists(string) , StoredProgram.GetVarValue(string) , StoredProgram.UpdateVariable(string, int) , StoredProgram.UpdateVariable(string, double) , StoredProgram.UpdateVariable(string, bool) , StoredProgram.DeleteVariable(string) , StoredProgram.IsExpression(string) , StoredProgram.EvaluateExpressionWithString(string) , StoredProgram.EvaluateExpression(string) , StoredProgram.Push(ConditionalCommand) , StoredProgram.Pop() , StoredProgram.Add(Command) , StoredProgram.NextCommand() , StoredProgram.Commandsleft() , StoredProgram.PC , ArrayList.Adapter(IList) , ArrayList.Add(object) , ArrayList.AddRange(ICollection) , ArrayList.BinarySearch(int, int, object, IComparer) , ArrayList.BinarySearch(object) , ArrayList.BinarySearch(object, IComparer) , ArrayList.Clear() , ArrayList.Clone() , ArrayList.Contains(object) , ArrayList.CopyTo(Array) , ArrayList.CopyTo(Array, int) , ArrayList.CopyTo(int, Array, int, int) , ArrayList.FixedSize(ArrayList) , ArrayList.FixedSize(IList) , ArrayList.GetEnumerator() , ArrayList.GetEnumerator(int, int) , ArrayList.GetRange(int, int) , ArrayList.IndexOf(object) , ArrayList.IndexOf(object, int) , ArrayList.IndexOf(object, int, int) , ArrayList.Insert(int, object) , ArrayList.InsertRange(int, ICollection) , ArrayList.LastIndexOf(object) , ArrayList.LastIndexOf(object, int) , ArrayList.LastIndexOf(object, int, int) , ArrayList.ReadOnly(ArrayList) , ArrayList.ReadOnly(IList) , ArrayList.Remove(object) , ArrayList.RemoveAt(int) , ArrayList.RemoveRange(int, int) , ArrayList.Repeat(object, int) ,

ArrayList.Reverse() , ArrayList.Reverse(int, int) , ArrayList.SetRange(int, ICollection) , ArrayList.Sort() , ArrayList.Sort(IComparer) , ArrayList.Sort(int, int, IComparer) , ArrayList.Synchronized(ArrayList) , ArrayList.Synchronized(IList) , ArrayList.ToArray() , ArrayList.ToArray(Type) , ArrayList.TrimToSize() , ArrayList.Capacity , ArrayList.Count , ArrayList.IsFixedSize , ArrayList.IsReadOnly , ArrayList.IsSynchronized , ArrayList.this[int] , ArrayList.SyncRoot , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MyStoredProgram(ICanvas)

Creates a new program storage linked to a drawing canvas

```
public MyStoredProgram(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas where all the drawing will happen

Properties

Commands

Gets a list of all the commands in this program Read-Only access to see all the commands available

```
public List<ICommandHandler> Commands { get; }
```

Property Value

List <ICommandHandler>

Methods

AddCommand(ICommandHandler)

Adds a new command to the program's list

```
public void AddCommand(ICommandHandler command)
```

Parameters

`command` ICommandHandler

The command to be added (circle, drawto, etc...)

ResetProgram()

Clears all the commands and resets the canvas to its blank state Useful when starting a new program form the beginning

```
public void ResetProgram()
```

Run()

Executes all the commands in the program one by onw Runs through the entire command list top to bottom and performs each operation

```
public void Run()
```

Exceptions

CommandException

Thrown if any command fails during execution

Namespace BOOSEapp.commands

Classes

CircleCommand

Handles the circle command - It draws a circle at current position Usage: circle 50

DrawToCommand

Handles the drawto command - draws a line from current position to a new position Usage: drawto 110,150

PenColourCommand

Handles the pen command - changes the pen colour Usage: pen 255,0,0 (for Red colour)

RectCommand

Handles the rect command - draws a rectangle at current position Usage: rect 50,100

Class CircleCommand

Namespace: BOOSEapp.commands

Assembly: BOOSEapp.dll

Handles the circle command - It draws a circle at current position Usage: circle 50

```
public class CircleCommand : ICommand, ICommandHandler
```

Inheritance

object ← CircleCommand

Implements

ICommand, ICommandHandler

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CircleCommand(ICanvas)

Creates a new Circle command with the drawing canvas

```
public CircleCommand(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas to draw on

Methods

CheckParameters(string[])

Checks if the circle parameters are vaild - Needs exactly 1 number (radius)

```
public void CheckParameters(string[] parameters)
```

Parameters

`parameters` string[]

Should have 1 value: [radius]

Exceptions

CommandException

Thrown if wrong number of parameters is given

Compile()

parses and validates the radius - makes sure it is a positive number

```
public void Compile()
```

Exceptions

CommandException

Thrown if invalid or negative radius is given

Execute()

Draws a circle centered at the current position with the specified radius The circle outline uses current pen colour and thickness

```
public void Execute()
```

Set(StoredProgram, string)

Sets up the command with the program and parses the parameters

```
public void Set(StoredProgram program, string parameterList)
```

Parameters

`program` StoredProgram

The program that this command belongs to

`parameterList` string

Srting with the radius value like "50"

Class DrawToCommand

Namespace: BOOSEapp.commands

Assembly: BOOSEapp.dll

Handles the drawto command - draws a line from current position to a new position Usage: drawto 110,150

```
public class DrawToCommand : ICommand, ICommandHandler
```

Inheritance

object ← DrawToCommand

Implements

ICommand, ICommandHandler

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DrawToCommand(ICanvas)

Creates a new DrawTo command with the drawing canvas

```
public DrawToCommand(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas to draw on

Methods

CheckParameters(string[])

Checks if the parameters are valid - requires 2 numbers exactly (x and y)

```
public void CheckParameters(string[] parameters)
```

Parameters

`parameters` string[]

Should have 2 values: [x , y]

Exceptions

CommandException

Thrown if wrong number of parameters is given

Compile()

Parses and validates the coordinates - makes sure they are positive numbers

```
public void Compile()
```

Exceptions

CommandException

Thrown if coordinates are invalid or negative

Execute()

Draws a straight line from current position to the target coordinates Uses the current pen colour and thickness for the line

```
public void Execute()
```

Set(StoredProgram, string)

Sets up the command with the program and parses the parameters

```
public void Set(StoredProgram program, string parameterList)
```

Parameters

`program` StoredProgram

The program that this command belongs to

`parameterList` string

String with comma seprated coordinates "150,200"

Class PenColourCommand

Namespace: BOOSEapp.commands

Assembly: BOOSEapp.dll

Handles the pen command - changes the pen colour Usage: pen 255,0,0 (for Red colour)

```
public class PenColourCommand : ICommand, ICommandHandler
```

Inheritance

object ← PenColourCommand

Implements

ICommand, ICommandHandler

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PenColourCommand(ICanvas)

Creates a new PenColour command with the drawing canvas

```
public PenColourCommand(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas to draw on

Methods

CheckParameters(string[])

Checks if the parameters are valid - needs exactly 3 numbers (red, green , bluw)

```
public void CheckParameters(string[] parameters)
```

Parameters

`parameters` string[]

Should have 3 values: [Red, Green, Blue]

Exceptions

CommandException

Thrown if wrong number of parameters is given

Compile()

Parses and validates the colour values - Makes sure they are in between 0-255

```
public void Compile()
```

Exceptions

CommandException

Thrown if parameters are less than 0 or greater than 255

Execute()

Changes the drawing colour to the specified RGB value given Affects all future drawings until the colour is changed again

```
public void Execute()
```

Set(StoredProgram, string)

Sets up the command with the program and parses the parameters

```
public void Set(StoredProgram program, string parameterList)
```

Parameters

`program` StoredProgram

The proggram this command belongs to

`parameterList` string

String with comma seprated RGB values "255,255,0"

Class RectCommand

Namespace: BOOSEapp.commands

Assembly: BOOSEapp.dll

Handles the rect command - draws a rectangle at current position Usage: rect 50,100

```
public class RectCommand : ICommand, ICommandHandler
```

Inheritance

object ← RectCommand

Implements

ICommand, ICommandHandler

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

RectCommand(ICanvas)

Creates a new Rect command with the drawing canvas

```
public RectCommand(ICanvas canvas)
```

Parameters

`canvas` ICanvas

The canvas to draw on

Methods

CheckParameters(string[])

Checks if the parameters are valid - needs exactly 2 positive numbers (width and height)

```
public void CheckParameters(string[] parameters)
```

Parameters

`parameters` string[]

Should have 2 values: [Width and Height]

Exceptions

CommandException

Thrown if wrong number of parameters is given

Compile()

Parses and validates the rectangle dimensions - makes sure they are positive numbers

```
public void Compile()
```

Exceptions

CommandException

Thrown if Height/Width parameters are invalid or not positive numbers

Execute()

Draws a rectangle starting from the current position Uses current position a top-left corner, extends by width and height

```
public void Execute()
```

Set(StoredProgram, string)

Sets up the command with the program and parses the parameters

```
public void Set(StoredProgram program, string parametersList)
```

Parameters

`program` StoredProgram

The program this command belongs to

`parametersList` string

String with comma seprated dimensions "100,50"