# Lab 7 – EntityJig

In this lab you will create a new class that inherits from the EntityJig class. When a class is inherited from EntityJig two methods need to be implemented. These are the Sampler and Update. In steps 1-32 you create the class. The rest of the lab (steps 33-40) you will add code to an existing command named “circleJig” that uses the class to prompt the user for the center and radius of the circle.

The comments are the steps for the lab. You can copy the comments and code below to the vb file in your existing project (need to copy inside of an existing class) or open the Lab7 project that already has the procedure “CircleJig” and these steps.

' Start of Lab7

' 1. Create a Class named MyCircleJig that Inherits from EntityJig.

' The EntityJig class allows you to "jig" one entity at a time.

' When you use the Inherits keyword, VB.NET automatically inserts the two

' functions that must be overriden. These are Sampler which is used to get

' input from the user and Update which is used to update the entity that is

' being jiged. In this lab we are using a Jig to create a circle.

' Note: Put the End Class after step 33.

' Move the start of the Sampler Function below step 6. Move the "End Function" below step 28.

' Move the update function below step 28 and the "End Function" below step 33. (above "End Class")

' 2. We need two inputs for a circle, the center and the radius.

' Declare a Private member varaiable as a Point3d named

' "centerPoint" and a private member variable as a Double named "radius".

' 3. Because we are going to have 2 inputs, a center point and a radius we need

' to keep track of the input number. (used to determine which value we are getting).

' Declare a private member variable as a Integer named "currentInputValue".

' 4. We will use a Property to get and set the variable created in step 3.

' (This value is accessed outside of the class). Declare a Property named

' "CurrentInput() as an Integer. The Get and Set will automatically be

' added by VB.NET. In the Get statement Return the member variable "currentInputValue"

' In the Set statement make the member variable "currentInputValue" equal to

' the value that is passed in.

' 5. Create the default constructor. Use Sub New and use ByVal with an Entity

' variable named ent.

' Note: Put the "End Sub" below step 6.

' 6. Call MyBase.New() and use the ent passed into the constructor. This needs

' to be done because the base class "EntityJig" does not have an accessible

' Sub New' that can be called with no arguments

' 7. (This step is in the Sampler function) Create a Select Case.

' For the case use the currentInputValue member variable,

' Note: Move the End Select after step 28

' 8.Use 0 (zero) for the case. (getting center for the circle)

' 9. Declare a variable named oldPnt as Point3d and instantiate it

' using the centerPoint member varible. This will be used to test to

' see if the cursor has moved during the jig. If the user does

' not change anything, Autocad continually calls the sampler function,

' and the update function, you will get a flickering effect on the screen.

' 10. Declare a variable as a PromptPointResult. Name it something like

' jigPromptResult. Instantiate it by making it equal to the return value of

' the AcquirePoint method of the JigPrompts oject passed into the function.

' Use something like "Pick center point : " for the message argument.

' 11. Check the status of the PromptPointResult created in step 11. Use the

' Status property and check if it is equal to PromptStatus.OK in an "If Then"

' statement.

' Note: Put the "End If" after step 14

' 12. Make the centerPoint member variable equal to the Value

' property of the PromptPointResult created in step 11

' 13. Check to see if the cursor has moved. Use an "If Then" Statement

' in the test use the DistanceTo property of the Point3d variable created

' in step 9. For the Point3d argument use the centerPoint variable. Use

' less than "<" and see if is smaller than 0.0001

' Note: put the "End If" after step 14

' 14. If we get here then there has not been any change to the location

' Return SamplerStatus.NoChange

' 15. There has been a change in the location so Return the SamplerStatus.OK

' 16. Use 1 for the case. (getting radius for the circle)

' 17. Declare a variable named oldRadius as double and instantiate it

' using the radius member varible. This will be used to test to see if

' the cursor has moved during jigging for the radius.

' 18. Declare a variable as a JigPromptDistanceOptions. Name it something like

' jigPromptDistanceOpts. Instantiate it by making a New JigPromptDistanceOptions.

' For the Message argument use something like "Pick radius : "

' 19. Make the UseBasePoint property of the JigPromptDistanceOptions created

' in step 19 True

' 20. Make the BasePoint property of the JigPromptDistanceOptions created

' in step 19 equal to the centerPoint member variable

' 21. Now we ready to get input. Declare a vaiable as PromptDoubleResult

' name it something like "jigPromptResult". Instantiate it using the

' AcquireDistance method of the JigPrompts passed into the Sampler function.

' Pass in the JigPromptDistanceOptions created in step 18.

' 22. Check the status of the PromptDoubleResult created in step 22. Use the

' Status property and check if it is equal to PromptStatus.OK in an "If Then"

' statement.

' Note: Put the "End If" after step 27

' 23. Make the radius member varialble equal to the Value

' property of the PromptDoubleResult created in step 22

' 24. Check to see if the radius is too small Use an "If Then" Statement

' in the test use the System.Math.Abs() For the Double argument use the

' radius member variable. Use less than "<" and see if it is smaller than 0.1

' Note: put the "End If" after step 25

' 25. Make the Member variable radius = to 1. This is

' just an arbitrary value to keep the circle from being too small

' 26. Check to see if the cursor has moved. Use an "If Then" Statement

' in the test use the System.Math.Abs() method. For the Double argument

' subtract the radius member variable from the oldRadius. Use

' less than "<" and see if is smaller than 0.001

' Note: put the "End If" after step 27

' 27. If we get here then there has not been any change to the location

' Return SamplerStatus.NoChange

' 28. Return the SamplerStatus.OK

' 29. In this fucntion (Update) for every input, we need to update the entity

' Create a Select Case. For the case use the currentInputValue member variable,

' Note: Move the End Select after step 33.

' 30. Use 0 (zero) for the case. (Updating center for the circle)

' 31. The jig stores the circle as an Entity type. Change it to a circle

' so we can access the properties easily. Use CType(). For the Expression

' argument use Me.Entity and for the Type use the Circle. Make the Center

' property equal to the centerPoint member variable

' 32. Use 1 for the case. (Updating radius for the circle)

' 33. The jig stores the circle as an Entity type. Change it to a circle

' so we can access the properties easily. Use CType(). For the Expression

' argument use Me.Entity and for the Type use the Circle. Make the Radius

' property equal to the radius member variable

' Note: continue to step 34 in circleJig Sub below

' create a command to invoke the EntityJig

<CommandMethod("circleJig")> \_

Public Sub CircleJig()

' 34. Create a new instance of a circle we want to form with the jig

' Declare a variable as a Circle named circle. Instantiate it by making

' it equal to a New Circle. For the center use Point3d.Origin. For the

' normal use Vector3d.ZAxis. Make the Radius 10.

' 35. Create a new jig. Declare a variable as a a New MyCircleJig.

' (the name of the class created in steps 1-32). Pass in the

' Circle created in step 34

' 36. Now loop for the inputs. 0 will be to position the circle and 1 will

' be to set the radius. Use a For loop with two iterations.

' For i As Integer = 0 To 1

' Note: Put the Next statement after step 41.

' 37. Set the current input to the loop counter. Use the CurrentInput

' property of the class variable created in step 35. (make it equal to i)

' 38. Get the editor object. Declare a variable as Editor and instantiate it

' with the Editor property of the MdiActiveDocument.

' 39. Invoke the jig. Declare a PromptResult variable and instantite it by

' making it equal to the return of the Drag method of the Editor created

' in step 38. Pass in the MyCircleJig class created in step 35.

' 40. Make sure the Status property of the PromptResult variable created in

' in step 39 is ok. Use an "If Then" statement. For the test see if the

' promptResult.Status is equal to PromptStatus.Cancel Or PromptStatus.Error.

' Note: Put the "End If" after step 41

' 41. some problem occured. Return

' End of Lab 7.

' Note: If you named your Circle variable something other than

' "circle" then change the code below to reflect this. It adds

' the circle to the database.

' once we are finished with the jig, time to add the newly formed circle to the database

' get the working database

Dim dwg As Database = Application.DocumentManager.MdiActiveDocument.Database

' now start a transaction

Dim trans As Transaction = dwg.TransactionManager.StartTransaction

Try

' open the current space for write

Dim currentSpace As BlockTableRecord = trans.GetObject(dwg.CurrentSpaceId, OpenMode.ForWrite)

' add it to the current space

currentSpace.AppendEntity(Circle)

' tell the transaction manager about it

trans.AddNewlyCreatedDBObject(Circle, True)

' all ok, commit it

trans.Commit()

Catch ex As Exception

Finally

' whatever happens we must dispose the transaction

trans.Dispose()

End Try

End Sub