# 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 1-38 you create the class. The rest of the lab (steps 39-46) 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 cs file in your existing project or open the Lab7 project that already has the procedure “CircleJig” and these steps. If you copy these steps then put the command “circleJig” in the class with the other commands.

// Start of Lab7

// Note: This project will not compile until the lab steps are completed.

// There is code in the CircleJig function that needs objects created

// by the lab.

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

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

// When you inherit from EntityJig you need override two functions.

// 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 closing curley brace after step 38.

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

// Declare a Private member varaiable of the class 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 int 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 int property named

// CurrentInput{}. Use get and set to return the member variable "currentInputValue"

// Use a set statement to make the member variable "currentInputValue" equal to

// the value that will set in when this class property is accessed.

// 5. Create the default constructor. Pass in an Entity variable named ent.

// Derive from the base class and also pass in the ent passed into the constructor.

// 6. Override the Sampler function. Use the protected keyword and it returns

// Autodesk.AutoCAD.EditorInput.SamplerStatus. Use Autodesk.AutoCAD.EditorInput.JigPrompts

// for the single argument. Name the argument "prompts"

// Note: put the closing curley brace below step 29.

// 7. (This step is in the Sampler function) Create a switch statement.

// For the case use the currentInputValue member variable,

// Note: Move the closing curley brace 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

// if the test is not done.

// 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 10. Use the

// Status property and check if it is equal to PromptStatus.OK in an "if"

// statement.

// Note: Put the closing curley brace after step 14.

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

// property of the PromptPointResult created in step 10

// 13. Check to see if the cursor has moved. Use an "if" 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 closing curley brace after step 14

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

// return SamplerStatus.NoChange

// 15. If the code gets here than there has been a change in the location so

// return 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 18 True

// 20. Make the BasePoint property of the JigPromptDistanceOptions created

// in step 18 equal to the centerPoint member variable

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

// Name it something like "jigPromptDblResult". 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 21. Use the

// Status property and check if it is equal to PromptStatus.OK in an "if"

// statement.

// Note: Put the closing curley brace after step 27

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

// property of the PromptDoubleResult created in step 21

// 24. Check to see if the radius is too small Use an "if" 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 closing curley brace 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" 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 closing curley brace after step 27

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

// Return SamplerStatus.NoChange

// 28. If we get here the cursor has moved. return SamplerStatus.OK

// 29. Return SamplerSataus.NoChange. This will not ever be hit as we are returning

// in the switch statement. (just avoiding the compile error)

// 30. Override the Update function. Use the protected keyword. The function

// returns a boolean and does not have any arguments

// Note: put the closing curley brace below step 38

// 31. In this function (Update) for every input, we need to update the entity

// Create a switch statement. For the case use the currentInputValue member variable,

// Note: Move the closing curley brace after step 37.

// 32. Use 0 (zero) for the case. (Updating center for the circle)

// 33. The jig stores the circle as an Entity type. Cast it to a circle

// so we can access the properties easily. Use this.Entity and for the

// cast use the Circle class. Make the Center property equal to the

// centerPoint member variable

// 34. break out of the switch statement

// 35. Use 1 for the case. (Updating radius for the circle)

// 36. The jig stores the circle as an Entity type. Cast it to a circle

// so we can access the properties easily. Use this.Entity and for the

// cast use the Circle class. Make the Radius property equal to the radius

// member variable.

// 37. break out of the switch statement

// 38. Return true.

// Note: continue to step 39 in circleJig function above

// create a command to invoke the EntityJig

// Note: This needs to be in the class with the other commands.

// (Not part of the “CircleJig” class created in stesp 1-38)

[CommandMethod("circleJig")]

public void CircleJig()

{

// 39. 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.

// 40. Create a new jig. Declare a variable as a a new MyCircleJig.

// (the name of the class created in steps 1-38). Pass in the

// Circle created in step 39

// 41. 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 (int i = 0; i <= 1; i++)

// Note: Put the closing curley brace after step 46.

// 42. Set the current input to the loop counter. Use the CurrentInput

// property of the class variable created in step 40. (make it equal to

// the counter variable in the loop. (will be either 0 or 1)

// 43. Get the editor object. Declare a variable as Editor and instantiate it

// with the Editor property of the MdiActiveDocument.

// 44. 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 43. Pass in the MyCircleJig class created in step 40.

// 45. Make sure the Status property of the PromptResult variable created in

// in step 44 is ok. Use an "if" statement. For the test see if the

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

// Note: Put the closing curley brace after step 46

// 46. some problem occured. Return

// End of Lab 7.

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

// "circle" then change the code below which adds the circle to

// the database to reflect this.

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

// get the working database

Database dwg = Application.DocumentManager.MdiActiveDocument.Database;

// now start a transaction

Transaction trans = dwg.TransactionManager.StartTransaction();

try

{

// open the current space for write

BlockTableRecord currentSpace = (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 (Exception ex)

{

}

finally

{

// whatever happens we must dispose the transaction

trans.Dispose();

}

}