#### **Task 0: Adding the Point object to your project** **-----------------------------------------------**

Download point.[h, cpp] from canvas. Move these files into your plotproject\_starter folder via the command line or a file explorer. Important! To complete the exercises below, **do not** use the "Add Existing Files..." option in QtCreator. The objective of this exercise is for you to get a better handle on how QtCreator works.

1) Can you see these files under the "Headers" and "Sources" folder icons in Qt Creator?

2) Edit your .pro file to include the point.[h, cpp] files. What happens in Qt Creator when you edit these files?  
Hint: make sure to include "\" characters if necessary after previously listed files!

3) Add an include for "point.h" into your plotwindow.[h, cpp] files. Make sure that your project can still build.

#### **Task 1: Exploring Point** **-------------------------**

1) What object(s) does the Point object inherit from? Can a Point object emit a signal?

The QObject and QGraphicsItem

2) What are the static fields (class attributes) of the Point class? What do you think that they are for/will be for?

Get width and the width

They are getting the width of a point since they will not change

3) What are the non-static fields (class attributes) of the Point class?

X y color

4) What are the purposes of the following methods (go read the qt documentation and the documentation on our github in the examples folder):

a) boundingRect

this bounds the painting

b) shape

creates shapes that are spread over areas of the screen

c) paint

Qt's paint system enables painting on screen and print devices

**Task 2: Make sure you have axes**  
---------------------------------------------

// Calculate the mid points  
int x\_mid = view->frameSize().height() / 2;  
int y\_mid = view->frameSize().width() / 2;// draw the axes  
scene->addLine(0, x\_mid, view->frameSize().width(), x\_mid);  
scene->addLine(y\_mid, 0, y\_mid, view->frameSize().height());

Make sure that point.cpp and point.h are successfully incorporated.

Here are the basic equations that you'll use to calculate x and y adjusted to the gui layout so that a user could enter 0 for x and 0 for y and have the point get drawn in the middle of your QGraphicsScene:

int x\_adj = x + (ui->plotGraphicsView->frameSize().width() / 2);  
int y\_adj = (-1 \* y + (ui->plotGraphicsView->frameSize().height() / 2));

You may find that you need to adjust these slightly in your explorations today!

**Task 3: Adding a Point to our scene (by instantiating one in your PlotWindow constructor)**  
**-----------------------------------------------**

1) Instantiate a Point \* in your PlotWindow constructor (make sure to pass a QColor as the first parameter. A QColor can be instantiated by giving it 3 int values for red, green, blue).

Once you've done this, call the scene->addItem method passing in your Point \*. Make sure that when you run your program you can see your Point! (Hint: if you give your Point the coordinates 0, 0, it will appear in the upper left corner, as these are interpreted as 0, 0 in the gui layout.)

Here's an example QColor instantiation:

QColor color(255, 255, 255);

\* Make sure that you instantiate your Point as a pointer--the scene->addItem(QGraphicsItem \* item) method takes a pointer to a QGraphicsItem (which Point inherits from) as a parameter!\*

copy + paste your code here:

Point \* point = new Point(QColor(25.5, 41.2, 88.2), 0,0)

Scene->addItem(point);

X\_adj

Y\_adj

2) What lines of code do you need to write to add a Point to your scene that appears at the origin of your graph?

Point \* point = new Point(QColor(25.5, 41.2, 88.2),x\_adj, y\_adj )

**Task 4: Adding some Points to our scene (Add Point button)**  
**-----------------------------------------------**  
1) Create a slot that is triggered when the "*Add Point*" button is clicked. You can use t he default slot if you'd like by clicking "*go to slot*" from the UI or you can implement your own. If you implement your own, don't forget the connect statement that you'll need to add in the PlotWindow constructor!

2) Make this slot so that it actually adds a Point to your graphics scene. Use the following code (and add to it!) to get the coordinates from the text boxes as integers:

All of this goes in the slot that gets triggered by a click on the add point button.

// &safe is the address of a boolean value that will be set to True if the   
// method successfully converted the text to an int and False otherwise  
bool safe = false;  
int x = ui->xCoord->text().toInt(&safe);  
if (!safe) {  
return;  
}  
// do the same thing for the y coordinate

// then create your point and add it to the scene!

Make sure that the Points appear where the user thinks that they should appear!

*Hint: you'll want to account for the diameter of the Point itself! (take a look at the fields in the Point class).*

**Task 5: Emitting a Signal from a Point (custom everything!)**  
**-------------------------------------------------**  
1) Take a look at the Point header file. Add the following code to this file defining a custom signal and adding the mousePressEvent function.

Point.h

signals:  
 // for distance  
 void PointSelected(int x, int y);  
  
protected:  
 void mousePressEvent(QGraphicsSceneMouseEvent \*event) override;

Point.cpp

void Point::mousePressEvent(QGraphicsSceneMouseEvent \*event)  
{  
 qDebug() << "point clicked!";  
}

What parameters does the PointSelected signal take? What line of code would you write to emit this signal from the mousePressEvent method in Point?

2) Run your application. Has anything changed in terms of interaction? Why or why not?

3) When you add a Point to your scene in your PlotWindow, connect that Point and it's PointSelected signal to a new slot in your PlotWindow class. This new slot must have the same parameters as the signal that it is receiving. Implement this slot so that it prints the x and y coordinates of the clicked point via qDebug().

Hint: your new slot needs to be a custom slot, like SlotTest from the previous lecture.

4) Will signals ever have return values? why yes/ why no?

**Task 6: Making our Point change colors when we click it**  
**-----------------------------------------------**  
1) Finish implementing the void Point::mousePressEvent(QGraphicsSceneMouseEvent \*event) method in point.cpp by making it so that when a user clicks a point, it changes colors. This method is automatically called whenever a user clicks a Point. Copy + paste the code that you added here.