**Name: Jason Truong**

For each of the below questions, write a short sentence or two to express (in your own words) your answer. Keep the answers short, but use complete, correct, English sentences.

If it helps to clarify the questions, feel free to mentally prefix all the questions with the phrase "According to the video…"

* After you’ve watched all the videos, please answer this question:  
  Of all the videos that you watched, if you could pick one video to be re-recorded by the instructor outside of class which would you choose? Why?  
  (Keep in mind the recording outside of class will omit any pauses from the instructor answering student questions, have less hemming and hawing, etc, and generally be more concise)

|  |
| --- |
| < Write your answer here > |

**VIDEO: Passing an object to a method (objects as parameters)**

(It may be best to watch this video in the ‘fullscreen’ mode)

The video starts with an overview of the program that we’ll be using in this video; it’s great review up until about the 2:30 mark.

* What is the overall goal of passing an object to a method?

|  |
| --- |
| The goal is passing a 'point' object or two into a method. |

* Within this particular program, what is meant by the “leftmost” point?

|  |
| --- |
| On a standard cartesian x-y coordinate graph, the leftmost point would be the smallest value of x. |

* What pattern do you follow to pass a pair of points into a method (such as PrintLeftMost)?

|  |
| --- |
| You pass a type, then you give it a name, then do it twice.  public void PrintLeftMost(Point p1, Point p2) |

* On the call**ing** side, how do you pass (say) the **origin** and **near** points into the PrintLeftMost method?

|  |
| --- |
| After creating the PrintComparer object; comparer, you call it using  comparer.[whatever].  comparer.PrintLeftMost(origin, near); |

* Why do we call PrintLeftMost(near, far) then immediately call PrintLeftMost(far, near)

|  |
| --- |
| because the comparer objects in main are being called in the order of:  PrintLeftMost(near, far);  and then  PrintLeftMost(far, near); |

**VIDEO: Objects are passed (as a parameter) by reference**

* In the example project the program first prints out the starting value of the **near** point, then calls the MovePointLeftwards command, then prints out the ending value of the **near** point.  
  What is this code trying to test / what question will this program help us answer?

|  |
| --- |
| The code is trying to test if the original copy of the parameter is the same parameter or not. |

* What does “pass by reference” mean?

|  |
| --- |
| Passing a reference is taking the original object into the parameter and making changes applicable. |

* What does “pass by value” mean?

|  |
| --- |
| Passing by value is creating a copy and passing through the parameters with the copy instead of the original. |

* What are some of the advantages of passing something (an object, an array) by reference?  
  What are some of the disadvantages/dangers of passing something (an object, an array) by reference?

|  |
| --- |
| Pass of reference is more efficient but more dangerous since there's only one copy of the object in reference. |

**VIDEO: Return an object from a method (objects as return values)**

(Note that the new material starts around the 2:00 mark – you can skip ahead to that point if you want)

* Why is it bad to have your constructor ask the user for input?

|  |
| --- |
| The constructor should just initialize objects and not ask for user inputs, this is because of the console's print method is going to run into issues for the object's lifespan. |

* How do you declare (in C#) that your method will be producing/returning a Point object?

|  |
| --- |
| With the CreatePoint Method, it would first create two values to fill the method, then asks the user to fill in those values, afterwards it feeds the values into a constructor and calls it. |

* How do you tell C# exactly which Point object you wish to return when the method is executing?

|  |
| --- |
| You have to declare the return value in the method. Int, Method, Double, Array, Etc. |

* Why will you get an error when attempting to call pt2.Print() prior to the pt2 = ptMaker.CreatePoint(); line?

|  |
| --- |
| The error will happen because there is no value in pt2 to even print. |

* What is one very common error that you absolutely do not want to do?  
  Also make sure to explain briefly and intuitively why this is bad.

|  |
| --- |
| You don't want to put an object to create a new object with empty values pointing to nothing, and then being called at a space in the stack @ a memory address.  We overloaded the object, allocated the values into a new object, which overwrote the previous objects in value and in stack.  So the previous allocated memory @ that address is now garbage memory. |

* When the CreatePoint method returns **localPoint**, what is it actually returning?

|  |
| --- |
| The memory address the object was created at. |

* What is the term for a block of memory that has no references pointing to it?

|  |
| --- |
| Garbage memory. |

**VIDEO: Class with an array of Double values inside it (GradeTracker)**

* What is the major difference between the array that we previously looked at, and what we’re going to look at today?

|  |
| --- |
| Before we looked at an array that looked at each slot as an object. Now we're looking as an array as an object. |

* When creating a new GradeTracker object, what will it do (array-wise)?

|  |
| --- |
| You create an array that's 10 elements wide. |

* What will the PrintGrades method do?

|  |
| --- |
| Print all the slots in the array one at a time. |

* How do you declare an instance variable that is an array (of, say, doubles)? Where do you put that declaration?

|  |
| --- |
| You give the name of the type, and then you just give it an actual name.  double[] grades;  for example^. |

* Why do you NOT need to do the numGrades = 0; line within the constructor?

|  |
| --- |
| Because when a variable is initialized, the default variable of an integer variable is 0.  Thus, numGrades = 0. |

* Does the declaration (from the previous question) actually create the array of **double**s?   
  If not, what line of code actually creates the array (since line numbers aren’t being displayed in the video you can just copy the line of code into this quiz):

|  |
| --- |
| No.  grades = new double[10]; |

* Briefly, intuitively summarize how the AddGrade method will work.

|  |
| --- |
| AddGrade will utilize the numGrade variable and use the variable to find the next available space in the grades using the position of AddGrade, it would fill that position. It would then increment numGrade by one for the next time numGrade is used.  If the numGrade variable exceed the length of the grades.Array, it will skip the process entirely and finish the method. |

* In the PrintGrades method, why does the for loop iterate until i reaches numGrades instead of until I reaches grades.Length? What will this print out to the screen?

|  |
| --- |
| Because all the values after numGrades aren't recogonized as the user's desired grades to reveal.  At the current position, the numGrades is at 2.  Position slots are filled at [0] = 3.5, [1] = 1.7.  So it will print:  3.5  1.7  and stop before slot 2 cause the loop is desginated not to continue past 2. |

* In the example video, can you use the syntax **gt[0]** to access the grade in slot 0 within the GradeTracker object referred to by the **gt** variable?  
  (Why would it be bad to allow this?)

|  |
| --- |
| We can. It'd just be really bad because we'd have anyone modify it willy-nilly and change it however they want. Maybe even re-allocate it to a new array and ruin the system that way. |

(Again, make sure that you watch the remainder of the video so that you can develop an intuitive understanding of how the computer executes this code)

|  |
| --- |
| **Class with an array of Double values inside it (GradeTracker): Additional Demo Video**  There are no viewing quiz questions for this video.  This video explains the Histogram class which should save you time when working on the exercise that uses this class. |

**VIDEO: Class with an array of Object References inside it (PointTracker)**

* In this video we’re going to create an array of references to objects that is located where?

|  |
| --- |
| Have the array be in of itself inside of another object. |

* What does the PointTracker.Print method do? How many points will it print out the very first time that it’s called?

|  |
| --- |
| It essentially does the same function as the PointGrades.Print() from the previous video.  It will print as many before num, which in this case, print nothing. |

* Why is it very convenient to use the following line of code? How might it be superior to creating a local variable to hand to the Add method?  
  pt.Add( new Point( 10, 10 ) );

|  |
| --- |
| It creates a new Point object directly, it's more streamlined and can create a new object and immediately passing it in. So there's no extra variables, reducing line-count, and overall making it lossless. |

* In the following line of code, where should you NOT put a semi-colon?  
  (There are a lot of places where you shouldn’t put the semi-colon – make sure you list the one explained in the video because it’s a very common mistake that people make)  
  pt.Add( new Point( 10, 10 ) );

|  |
| --- |
| pt.Add( new Point( 10, 10 ) [<-- riiight here.] ); |

(Again, make sure that you watch the remainder of the video (after the 8 minute mark) so that you can develop an intuitive understanding of how the computer executes this code)

**VIDEO: Class with an array of Object References inside it WITH NULL VALUES (PointTracker)**

* What will this video deal with?

|  |
| --- |
| It's a culmination of all the videos we worked with now  Arrays in reference to an array to references of more objects. |

* What is different about this video?

|  |
| --- |
| Dealing with nulls! |

* How does the Print method differ from the one shown in the prior video?   
  What effect will this have on the running program?

|  |
| --- |
| It has an if statement to test for the position we're looking at if its null.  It'll help prevent crashing from null refrences. |

* Briefly, intuitively summarize (in your own words) how the new AddGrade method will work when you add the object referred to by the **origin** variable here:

|  |
| --- |
| The origin object creates a new Point object with point p as 0, and where as 0.  When the new pt.AddAt(origin, 0); takes the memory address of Point object origin, and uses that as the 'p' variable for what's passed. |