**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: OOP Basics**

* The basic idea behind object oriented programming is to combine what two things together, in one place?

|  |
| --- |
| Data and Behaviour |

* What is the first “part” of the rectangle class?

|  |
| --- |
| The first part are the private data fields. |

* Data should always be declared as what?  
  (Remember that you can always create a public method to access these data fields later.)

|  |
| --- |
| Private. |

* Even though the behavior (the methods) are defined for all instances of this class, calling the calculateArea method will cause the area to be calculated based on what?

|  |
| --- |
| The calculateArea() method will calculate based on the two Private-Doubles located in the same class it resides in.  Those variables aren't transient to the class, so thus remain with the methods to use. |

* What is the “constructor” used for?

|  |
| --- |
| It adjusts the permanent data of the class with the temporary data being used in the method. |

* How does the constructor copy a starting value for the width (or height) into the rectangle’s “permanent” memory?

|  |
| --- |
| It takes the temporary transient data called from the constructor-method's parameters and changes the values of the permanent data by equating them. |

* Given a choice between making a variable a permanent (“instance”) variable or a temporary variable (typically a local variable), which type should you prefer to use? Under what circumstances would you use the other one?

|  |
| --- |
| Favor the temporary variables over permanent variables, because of storage consumption. The alternative should only be necessary when we need a long-lived important variables. |

**VIDEO: OOP Encapsulation**

* In C# you can use either a getHeight() method or a C# property. Why will we be using the getHeight() method in this course?

|  |
| --- |
| Because its' concept plays nicely to other languages, like Java and C++. |

* In addition to allowing us to make basic changes to objects (such as changing **r1**’s height to be 10), what else does the accessor method make it easy for us to do?  
  (Hint: What if we try to change **r1**’s height to be **-**10)

|  |
| --- |
| It will help the program; if coded to, minimize the amount of 'bad data' that goes into the system. |

* If one makes the height field public what implications will that have for the program (in terms of who can access that field)?

|  |
| --- |
| Anybody, anywhere in the code, can modify the height. Whether we want them to or not. |

* Is the public/private/protected feature that we’re looking at a security mechanism?

|  |
| --- |
| Not at all. It's just part of the code. |

* Why are the constructor and calculateArea methods allowed to access the private **height** field?

|  |
| --- |
| They're able to access the private height variable is because they're all under the same class. |

* Can the **main**  method access the private **height** field of the **r1** object?

|  |
| --- |
| Nope! 'Protection-level difference'. |

* Why is it ok to call the calculateArea method outside the Rectangle class (say, in **main**)?

|  |
| --- |
| The method is classified as a public within the class, so thus it can be accessed outside of its designated class. |

* Let’s say that you wanted to ensure that the height field of all rectangle objects are not negative. Prior to the public/private/protected access-control mechanism, what was your only option for ensuring that this would be true? Why would this be difficult?

|  |
| --- |
| You just had to make sure to write your code very carefully.  The prudence needs to be shared amongst all your co-workers, before shenanigans kick in. |

* Fill in the blank: “Just to recap, encapsulation is:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”

|  |
| --- |
| You marking your data fields private or public. |

**VIDEO: OOP: (Instance) Methods, Object Allocation Demo**

* What is a good type of name for a class (and what are a couple of examples of bad names for a class)?

|  |
| --- |
| A category, or the type of things that would fall in the Class. Not like random names like "Bob" or "Steve". |

* If you wanted to add a “Print” command to a class named Car, where would you put the Print method’s definition? Describe the location in English, then copy the example code from the video (at about the 1:50 mark)

|  |
| --- |
| If you want to put the print command in the "Car" class, you would have to first make a method inside the Car class, which is:  class Car  {  public void Print()  {  Console.WriteLine("Car Object");  }  }  The method id called in definition within main with:  Car c  c.Print(); |

(After 2:15 the video answers a number of tangential questions –skim this if you’re interested, but don’t worry about recalling the details of this part of the video)

**VIDEO: OOP: Instance Variables**

* Fill in the blank: “Unless you’ve got a really good reason [to do otherwise], you should always make all your instance variables \_\_\_\_\_\_\_\_\_\_\_\_\_”

|  |
| --- |
| You should always make your instance variables 'private'. |

* When you create a new Car object, what value will be put into all of the instance variables for that particular object?

|  |
| --- |
| The default variable concerning the object. Integers would default at variable 0. Double would default at variable '0.0'. |

* Describe in your own words how the SetMaxSpeed method allows one to change the maximum speed for a car object.

|  |
| --- |
| The method takes a parameter from the main method, and takes the number taken from the parameter. |

* Why is it ok to set maxSpeed in the SetMaxSpeed method, and then use it in the Print method?

|  |
| --- |
| Because the methods are in the same Class as the private variables |

* How does writing out as your variable as **this.**maxSpeed help to make your program more clear?

|  |
| --- |
| 'this.' would just really hammer home the literal same variable in the same class. |

(Starting at about 9:30 there’s a very visual explanation of how memory is managed when creating and using objects. Even though there aren’t a ton of questions about this, definitely watch this in order to develop an intuition about how object creation & usage works!!)

* What is the purpose of the invisible parameter named **this**? What information does it pass from main to the SetMaxSpeed method?

|  |
| --- |
| The this. invisible parameter is holding the information of the same object and value its referencing to. It's the program's own way of understanding 'context'. |

* Inside the SetMaxSpeed method, how does C# figure out what the (variable) name maxSpeed refers to?

|  |
| --- |
| It checks for a local variable, and then a parameter of the same name, then utilizes the invisible 'this' to find the variable in the class. |

* Because of the procedure you outlined in your answer to the prior question, is it possible to create a local variable that has the same name as an instance variable?

|  |
| --- |
| There's nothing wrong with it. So its valued c# code. |

* Should you ever create a local variable that has the same name as an instance variable?   
  ( Hint: NO!!!!!!!!!!!!!! )

|  |
| --- |
| No, because its visually confusing. |

* Does the method’s name cause the action/effect of the method?   
  If not, why is it still a good idea to choose method names that describe the each method does?

|  |
| --- |
| No, the name doesn't effect the code's behaviour.  It's just for extreme clarity. Like 'set' is common courtesy to do so for anything that would set up a variable. |

* The “Stack” is very, very efficient at doing what?

|  |
| --- |
| Allocating and Deallocating. |

**VIDEO: OOP: Access control (public/private), getter/setter methods**

* The primary purpose of access control is prevent my co-workers from doing what?

|  |
| --- |
| Making edits to certain portions of the code. |

* What can the SetMaxSpeed method do that will help prevent errors for being made in the overall program?

|  |
| --- |
| Prevent any of the max speed to be negative |

* What does the keyword “private” mean / what does it do?

|  |
| --- |
| It locks any object or method to the class it resides in. |

* If you try to access a private data member (say, in **main**), what error message will you get?

|  |
| --- |
| You'll get an error regarding 'security level'. |

* Why is **main** allowed to call SetMaxSpeed? Why is SetMaxSpeed allowed to change **maxSpeed**?

|  |
| --- |
| Because SetMaxSpeed resides in the same Class as the variable maxSpeed |

* Another advantage to forcing everyone to use accessor methods is that you can change what part of the class/program and yet NOT have to change the rest of the program?

|  |
| --- |
| Change anything internal in the class, like the Speeds without actually needing to change them. |

(Around the 9 minute mark there’s a blurb about exception handling – it is interesting to know about, but you’re not required to know it (unless there’s other material in this class that specifically requires you to know it) ).

* Does access control protect you from intentionally malevolent actions by your co-workers? If not, then where is it useful?

|  |
| --- |
| No, they can look at the source code.  This is a 'protection' for important values in the code that needs to remain private. |

**VIDEO: Overloading**

* Are over**load**ing a method and over**riding** a method the same thing? If not, give a really quick, intuitive explanation of each one (you haven’t seen overriding yet so don’t worry about providing a detailed/entirely correct explanation – the main thing is that you know these are different)

|  |
| --- |
| No.  Overiding is based on a separate 'polymorphism' topic we will cover later in the cover.  Overloading is when you have multiple methods of the same name with different parameters. One method could have more parameters than another. |

* There’s an example of overloading in the video involving a printBox method (right around the 3 minute mark) – briefly summarize what we were trying to accomplish with the printBox methods, and why it’s useful to have multiple methods that all have the same name.

|  |
| --- |
| The situation where you have either one-many parameters or one-few than what you're pushing, it will be prepared for both situations. |

* The basic rule for figuring out if you’ve legally overloaded a method is if you’ve got enough information at what point in the program? What information do you need, specifically?

|  |
| --- |
| Which version to call with what information you typically have available. |

* Fill in the blank: “Overloading makes sense when you have one \_\_\_\_\_\_\_\_\_\_\_, but a whole bunch of different \_\_\_\_\_\_\_\_\_\_\_\_”

|  |
| --- |
| Action  Ways to call it |

* Can C# differentiate which version of the method to call based on the return value?  
  If not, why not?

|  |
| --- |
| No.  Because the program isn't required to actually work with the returned value, it sometimes just won't. |

* Is it less efficient to call one overloaded version of the function, and then have it immediately call the ‘real’ version? If so, why do we do it?

|  |
| --- |
| Yes. It's made for convenience and also mass-usage. |

* Give an example of an overloaded method that’s built into the .Net Framework Class Library:

|  |
| --- |
| In Console.cs there's 13 different versions of the WriteLine command. |

**VIDEO: OOP: Constructors**

* Implementing a constructor for your class will help you avoid making what sort of error?

|  |
| --- |
| Forgetting a default value. |

* When does the constructor method run/execute?

|  |
| --- |
| When the method is called and the parameter is passed to construct the object. |

* When the Car class has only 1 constructor, that takes a single parameter, what sort of compile-time can occur? Why is this error good / useful?

|  |
| --- |
| Forces people to give people an initialized value, so co-workers can follow your line of thinking. Needing a sensible default value. |

* What two things identify a method as being a constructor?

|  |
| --- |
| A constructor has the same name as the class and no return value. |

* How can you have one constructor call a different constructor? Give a brief snippet of C# code that demonstrates this, and then provide a brief explanation (in English, in your own words) about how this works:

|  |
| --- |
| Yes.  public Car(int m5)  {  maxSpeed = m5  }  public Car() : this(50)  {  //Car(50);  }  When the code of Car() is called in Main, it throws the default variable from the Car constructor and then moves it into the constructor that fits the parameters. |

**VIDEO: Basic Array of Objects**(You can jump to the 1:30 mark & watch from there – the first minute is not useful )

* In your own words BRIEFLY outline what the **new** operator does when asked to create an array of Demo objects. Make sure that you’re clear what value (if any) that gets placed into each array slot.

|  |
| --- |
| Referencing to: Demo[] ds = new Demo[5];  The new array allocates into the heap, and occurs at a memory adress unknown to us.  The entire array is replaced with 'null' in each slot in the array. |

* What does the keyword **null** mean/represent in C#?

|  |
| --- |
| 'null' means nothing. It's literally nothing, non-existant, not even zero - absence of it all. |

* What is a simple, intuitive rule for remembering what default value will be filled into a new array?

|  |
| --- |
| If its a data-type, just assume its going to be one fo the corresponding primitive data-types. |

* In the below snippet of C# code, what does line **(1)** do? What does line **(3)** do? Do you really need both? How are they different?

Demo [] ds = new Demo[ 5 ]; **(1)**  
Console.WriteLine( ds[ 0 ] );  
ds[1] = new Demo(); **(3)**

|  |
| --- |
| Take a new array of 5 spaces, and fills them all with 'null'.  It's going to make a new Demo object, and take the memory address it resides in, then fills the space of that array; in this case 1, and fills the space with the address value. |

* How do you call the printBox() method on the object at array slot #1 in the array named ds?

|  |
| --- |
| ds[1].printBox(); |

**VIDEO: Null References Within An Array**

* What happens when you attempt to call the printBox() method on the third slot in the ds array (ds[2]) ? Why does that happen?

|  |
| --- |
| It accesses the "Null" and then promptly crashes because there is 'null' there. A null reference point exception. |

* What sort of exception will see in when you attempt to call the printBox() method on the third slot in the ds array (ds[2]) ?

|  |
| --- |
| Null reference exception pointer |

* How can you check (using C# code) if an array slot is **not** null? Provide a short snippet of C# code, and a brief, intuitive explanation of how it works:

|  |
| --- |
| if (ds[2] != null); to check if the slot is not null. Looks into the array and checks on that slot if its null in there. |

* How can you check (using C# code) if an array slot **is** null? Provide a short snippet of C# code, and a brief, intuitive explanation of how it works:

|  |
| --- |
| if (ds[2] == null); to check if the slot in the array is null, it will go into that slot in the array and check if it is null. |