*LINQ Compare and contrast declarative and imperative programming in LINQ: Declarative programming* is when the programmer specifies the conditions that selected elements must satisfy such as in LINQ queries. This expresses *what* the program should accomplish without prescribing *how* to do it in terms of sequences of actions to be taken. *Imperative programming* is when the programmer uses repetitive statements (to filter arrays), which focus on the *steps* required to get the results. Imperative programming is the use of loops and if statements to cycle through the information and sort the necessary elements. ;

*Under What circumstances does the compiler provide a default constructor and how does it initialize instance variables?* The default constructor that takes no parameters is provided when it is invoked if the programmer does not explicitly provide one of their own. Default constructors are invoked whenever an object is instantiated using the **new** operator and no arguments are provided to **new.** The default is different than parameterless because;

*Memory leaks are common in C and C++ by less likely is C#. Why?* The Memory Leaks occurring during garbage collection and destruction process, when the references to the object that manages the resource are lost before the resource can be explicitly released, and it can no longer be released. This is less likely is C# because it has the garbage collector built in and it will automatically reallocate the memory so it is not lost.;

*How are readonly instace variables different than named constants?* The principle of least privilege states that code should be given the least amount of privilege and access as needed. Readonly specifies that an instance variable if an object is not modifiable and that any attempt to modify it after the object is constructed as an error. Const and readonly variables are both in all caps. A readonly instance variable does not become unmodifiable until after the constructor completes execution. Consts must be assigned at compile time whereas readonly variables must be assigned at run time. PUblic readonly int JAKE; Public const int JAKE;

*We have considered different ways to reuse code. Compare and contrast copy and paste, composition, and inheritance.* Copy and pasting code is a bad way to reuse code because if you change something in one place you must change it everywhere which is time consuming and inefficient. Composition (HAS –A) – a class can have references to objects of other classes as members. Public Class1 FirstName; public TimeClass Oclock; This is the best way to reuse code for smaller projects because it allows you to use the methods and objects of other classes so you do not need to have them in your class file. Inheritance (IS-A) is when the derived class inherits the members from base class (Derived class : base class) in order to use the methods, variables, etc, from the base class. This is a form of polymorphism. This is the best way to reuse code for large projects because it allows you to write general code in the base classes and then use that code inside the derived classes instead of rewriting the same code over and over;

*Compare and contrast protected and private access. Which do you prefer and why?* The Private access allows the derived class to inherit the base class’s members but are not able to directly access them by derived-class methods and properties. The protected access allows the children of the base class to use the base class’s members directly so I prefer protected. All non private base class members retain their original access modifier when they become members of the derived class (public, protected);

*Discuss the ways in which inheritance promotes software reuse, saves time, and helps prevent errors -Inheritance* allows for polymorphism, which is coding in general so the program can use the code in numerous places. This means the programmer is not typing the same thing in many different files or locations. This saves time by reducing redundancy and prevents errors by reducing the amount of actual typing being done;

*What is an abstract class? How would one typically be used in creating a hierarchy?* Abstract keywords indicate that a base class method can be overridden in derived classes. Abstract, in basic terms, means that the class is going to write general code (such as a method without a body) and use a matching method/property from the derived class and use that code to do the work. The abstract classes are usually the parent classes;

*Compare and contrast abstract classes with interfaces, which approach would allow completely unrelated classes to share a stucture? -* Abstract classes cannot be used to instantiate objects because they are incomplete. Abstract classes are too general to create real objects. These classes can be used to instantiate objects that are called concrete classes. Interfaces define and standardize the ways in which people and systems can interact with one another. An interface is typically used when unrelated classes need to share common methods so that they can be processed polymorphically. They describe a set of methods that can be called on an object to tell it to perform some task or return some information. Interfaces can only contain abtract methods, properties, indexers and events. (*Public interface IPayable{ })*. Interfaces offer the capability that requires unrelated classes implement a set of common methods. (A method that calculates a payment amount);

How does polymorphism enable you to program in the general? Discuss advantages of this - Polymorphism enables us to write applications that process objects that share the same base class in a hierarchy as if they were all objects of the base class. The polymorphism occurs when an application invokes a method through a base class variable at execution (run) time. The key advantages are extensibility: software that invokes polymorphic behavior is independent of the object types to which messages are sent; This allows for less coding, time and mistakes. ;

*A derived class can inherit interface or implementation from a base class. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation? -* An interface is typically used when unrelated classes need to share common methods so that they can be processed polymorphically. An interface is often used in place of an abstract class when there is no default implementation to inherit;

*What are abstract methods and describe when they are used -* Methods with the keyword, abstract in their declaration must be inside an abstract class. They do not provide implementations (Bodies) and implicitly virtual. The derived class will provide the implementation (if they wish to be concrete classes). This would be used when a method is created in a base class, such as abstract library item, and then implementation is performed in a derived class such as library book;

*How does polymorphism promote extensibility? -* Extensibility is when software that invokes polymorphic behavior is independent of the object types to which messages are sent; Polymorphism promotes extensibility: Software that invokes polymorphic behavior is independent of the object types to which messages are sent. New object types that can respond to existing method calls can be incorporated into a system without requiring modification of the base system. Only client code that instantiates new objects must be modified to accommodate new types. This is possible because a subclass object is a super class object as well. When invoking a method from that reference, the type of the actual referenced object, not the type of the reference, determines which method is called. A subclass reference can be aimed at a super class object only if the object is downcast;

*Describe the gui controls anchored and docked -* Anchored or docked means, on the gui, where and how the specific textbox/label/ button.. etc is located relative to the page. Anchored means that the item is at a fixed distance from the sides of the GUI form. Docked means that it is fixed to the top or bottom of the page and will span the entire length or height of the page;

*What is projection and give an example -* Projection is mostly used in LINQ statements. Projection refers to the operation of transforming an object into a new form that often consists only of those properties that will be used. By using projection, you can construct a new type that is built from each object. You can project a property and perform a mathematical function on it. You can also project the original object without changing it;

*Preconditions* and *postconditions* allow a programmer to specify **what** a method accomplishes *without* describing **how** the method accomplishes it so if a new developer comes in to make additions or changes to the old code it is easier to understand;

1. Discuss four ways in which you can assign base-class and derived-class objects to variables of base-class and derived-class types.
   1. *Explain the try –Catch – Finally blocks -* The try catch comes first and contains the code that could possibly throw an exception. The catch block looks for a specific or general exception and then performs some action such as printing a readable statement describing that exception. The catch blocks should be listed in specific or general exception order so by the last catch it will catch anything that may have fallen through. The finally block is guaranteed to execute regardless of whether an exception occurs. It is located after the catch and is great for releasing resources A catch or finally block MUST follow a try block. The finally block is guaranteed to execute regardless of whether an exception occurs. This makes the finally block ideal for code to release resources from the try block. This ensures that even if the program terminates due to an uncaught exception, the resource will be deallocated.;
2. Why is it considered bad form to handle errors by catching non-specific exceptions, such as **System.Exception** as a "catch all"? When would it be acceptable to use this approach?
3. Describe C#’s event-handling mechanism, including  
   (a) the *event sender* and *event handler*and  
   (b) the role of *delegates*in establishing which method(s) get called in response to a control’s event firing.
   1. The
4. Write a code fragment that creates an array of **LibraryItem** references named *libItems*. Allocate memory for the array to hold 10 references. Write a loop that steps through each object in the array and calls its **ToString** method, displaying its output to the console.
5. Write a code fragment that creates a **List** of **LibraryItem** references. Write a loop that steps through each object in the list and calls its **ToString** method, displaying its output to the console.
   1. List<LibraryItem> items = new List<LibraryItem>();
6. Write a code fragment that uses LINQ to select all values in the given array between 100 and 200 (inclusive) and put them in descending order.
7. Write a code fragment that uses LINQ to select all **LibraryBook** objects in the given list with a *CopyrightYear* that is from 2000 and beyond and put the books in ascending order by *CopyrightYear*.