Contents

[Day\_3 2](#_Toc71110538)

[Review 2](#_Toc71110539)

[OOP Pillars 3](#_Toc71110540)

[4 OOP 3](#_Toc71110541)

[Abstraction 3](#_Toc71110542)

[Interfaces 3](#_Toc71110543)

[Abstract classes 4](#_Toc71110544)

[Polymorphism 5](#_Toc71110545)

[Implementations of Polymorphism 5](#_Toc71110546)

[Inheritance 5](#_Toc71110547)

[Composition, Aggregation, Association 5](#_Toc71110548)

[Encapsulation 6](#_Toc71110549)

[Discussion 6](#_Toc71110550)

[Modifiers 7](#_Toc71110551)

[Abstract 7](#_Toc71110552)

[Const 7](#_Toc71110553)

[Read Only 7](#_Toc71110554)

[Read Only Vs. Const 7](#_Toc71110555)

[Sealed 7](#_Toc71110556)

[Static **Applied to: classes and class members** 7](#_Toc71110557)

[Virtual 7](#_Toc71110558)

[Override 8](#_Toc71110559)

[Discussion 8](#_Toc71110560)

Revature

# Day\_3

## Review

Written evaluation will be EOW (Friday)

What are the different/common BASH commands as opposed to Dotnet commands.

Q: What is DotNet?

A: DotNet is a development platform, A collection of languages , runtime that allows developers to develop cross platform. Managed code, managed by the CLR.

Q: What does it mean to be cross platform?

A: Allows code from other languages to compile with the use of Just In Time(JIT)

Q: What is the CLR?

A: Common Language Runtime, Similar to JVM

Q: What are Features of CLR?

A: Handles Runtime conversions, Converting it into a common language, Memory management using a garbage collector, as well as Exception handling.

Q: Why is memory management important?

A: Helps relieve space for more frequently used objects. Often checks if the object is being referenced and relieves space.

Q: What types of memory are we dealing with?

A: Heap and Stack;

Heap: Where all the current data is stored

Stack: Where all the strings, int, and other primitive variables, as well as other processes. Also stores references to the Heap.

Stack Trace: Shows what where your program failed.

Q: What is Language Interoperability ?

A: Capability of .NET to work with other .NET compliant languages. IE you could use C#, Visual Basic, and F# all on the same languages as they are all converted into a common language using the CLR.

You can use any languages, however, if they are not compliant they would need a wrapper.

Q: Whats the difference between value types and reference types?

A: Value type: Holds value directly. Stored in the Heap

Reference Type: Holds a reference to a location in memory, where that value is stored. Stored in the Stack

Q: What’s the CTS?

A: Common Type System: The common language for all .NET compliant languages.

Q: What happens if I use using against a non-referenced namespace

A: It looks if the project has been added, if not, it will check in the current namespace for the project, if not, it will look for the system area for the project.

# OOP Pillars

## 4 OOP

* “APIE”
  + Abstraction
  + Polymorphism
  + Inheritance
  + Encapsulation
* Each aspect rely slightly on other aspects of the pillars. (Overlapping)
* Key to learning these pillars; come up with the distinction of each pillar, as well as an example
  + Define them in your own words
* Coming up oral examination, held by QC, which we will find out who tomorrow 5/6
  + Looking for good wholistic review of concepts

## Abstraction

* “Avia Pervia” Make simple looks to look complicated – Massimo Bottura
* Ex. Your phone has many buttons that you know will have some functionality that you don’t know the underlying functionalities.
* A technique oh hiding complex details of an operation behind a functionality
* Would your end user even care?
* Implemented in C# using interfaces and abstract classes (as well as a User Interface(UI))

## Interfaces

* Group of related functionalities that a non-abstract class or a struct must implement
* Cannot be instantiated, therefore it cannot hold data
* Is both abstract and public
* Interfaces impose functionality/behaviour
* When Instantiated a concrete method using an interface, you can only access methods provided by the interface.
* Interface describes what the class behavior will act. Can be considered like an adjective to describe a noun.
* **Does not have constructors**, A key difference between abstract and interfaces.
  + This is due to the addition of default constructors.
* Wrapping functionality that goes together in a package

Q: Why would you use Interfaces as opposed to calling the class directly

A: Principle, good code needs to be Readable and Writeable. Your concrete classes must not rely on other concrete classes as part of good code. By Loosely coupling them, you can create features that are shared in a singular location.

## Abstract classes

* Denoted by the tag “Abstract”
* **Can have constructors**
* Can also have unimplemented methods as well as both abstract, and non-abstract methods.
* Abstract is used to impose/create hierarchy
* Abstract classes created structure among your classes
* Can be seen in the English language like nouns compared to proper nouns; George can be referenced as both the non-proper noun “Human” as well as the proper noun “George”.

## Polymorphism

* Functionality may change depending on the context
* Literally means “Many forms”
  + Present the same interface, but being utilized differently.
  + Ability to substitute different details for different needs

### Implementations of Polymorphism

* Ad hoc
  + Defines a common interface for an arbitrary set of individual types
  + Calling a class that inherits a class behavior
* Parametric
  + When the generic implementation can be the same, however the type is abstract.
    - Ex. List<E> implements; List<Int>,List<String>, List<Double>, etc. all add and hold the data types the same but are generalized in List<E>
* Inclusion
  + When a name denotes instances of many different classes under a superclass

Venkata’s Analogy of Family does a very good job of describing Polymorphism. Too long to type :S

## Inheritance

* Basing an object or class upon another object or class, retaining similar implementations
* EX. Main Menu = IMenu, using code reusable
* You can inherit from multiple interfaces but you can only inherit from one class(Abstract or Concrete), as long as two inherited classes share methods with the same name

## Composition, Aggregation, Association

<Slides are very self-sufficient here> If any issues, Phoebe provided [this](https://stackoverflow.com/questions/885937/what-is-the-difference-between-association-aggregation-and-composition) Stack Overflow

* Composition
  + Describes a relationship between two or more objects where one object maintains instances of the other objects.
* Aggregation
  + Relationship of two or more objects where one object contains multiple instances of other objects.
* Association
  + Relationship between two objects where one object uses a separate instance of another object as a part of its functionality.

## Encapsulation

Putting all your eggs into one basket

* Treat related data/behavior as a single unit/capsule
* Places Validation and processing of data into your data class
* Implemented in data hiding and wrapping
  + Hiding: accessing modifiers, enforcing your accessibility (Read, Write accesses)
  + Wrapping: Grouping logic in classes, assemblies, and namespaces)
* Access Modifiers: Define your accessibility of your data
  + Private
    - Only accessible within the same class
  + Public
    - Accessible by any artifacts regardless of where it is called
  + Protected
    - Only accessible by classes that derive from it
  + Internal
    - Only accessible within the same namespace
  + Protected Internal
    - Only accessible within the same namespace or any of its derived classes
  + Private Protected
    - Only accessible from derived classes within the same namespace

## Discussion

Q: What are the 4 OOP Pillars?

A: Abstraction, Polymorphism, Inheritance, Encapsulation

Q: Differences between Inheritance and Encapsulation?

A: Abstraction is to lower Complexity, Encapsulation is more Security.

Abstraction hides the implementation, Encapsulation limits access of the other classes to a particular object or class.

Q: What are some use cases for abstraction?

A: Use abstraction to force a more generic class to have a more specific implementation, You can’t implement a Restaurant and therefore must be more specific of what type of Restaurant.

Q: What are some good use Cases for Polymorphism?

A: Overloading Constructors so you don’t need to provide all the data at construction.

Generalizes implementation; Shapes Area is similar among things such as squares and rectangles.

# Modifiers

## Abstract

**Applied to: Classes and Class Members**

* Allows you to create classes that are incomplete and must be derived
* Cannot be in conjunction with static
* Implicitly virtual

## Const

**Applied to: Fields**

* Constant fields and locals aren’t variables and may not be modified
* Constants are;
  + Numbers
  + Bool
  + Strings
  + Null references
* Cannot be changed

## Read Only

**Applied to: Fields, Structs, and Structs Member**

* Values can only be set once, from then on, it can only be read

### Read Only Vs. Const

* Const are initialized at declaration
* Read only fields can be initialized at a later point
* Const Fields is a Compile-Time Constant
* Read Only can be used for run-time constants
* After initialization of a Read Only, Const and Read Only are relatively similar

## Sealed

**Applied: classes, class members**

* Prevents overriding methods

## Static **Applied to: classes and class members**

## Virtual

**Applied to: methods, prop, indexers, events**

* Declares that the method is overridable
* By default, methods are non-virtual
* Cannot be used with; static, abstract, private, or override modifiers.
* Ex. ToString() method

## Override

**Applied to: methods, prop, indexers, events**

* Allows the modification of virtual methods

## Discussion

Q: what are modifiers? Are they only limited to access modifiers?

A: Access Modifiers: Provide different levels of access to different elements of your code

Modifiers: Control how certain elements behave, and what your allowed to do with them

Q: Whats the difference between Const. and ReadOnly

A: ReadOnly can be instantiated later in the code while Const. are determined at definition.

Q: How do I make something overridable? How do I override it?

A: By declaring it “Override” to make it overridable. Also by declaring that it is virtual.

# Variance

**A form of Polymorphism**

## Type Variance

* What may be substituted
* Covariance:
  + Setting the type to a with a subclass relationship
* ContraVariance
  + The inverse of Covariance
* Invariance
  + When only that type is acceptable and not any super or sub types

## Type Casting

* Boxing
  + Upcasting
  + Implicit Type casting
  + EX. Covariance, Assigning a general type to a more specific type
* Unboxing
  + Downcasting
  + Obj A to int, must explicitly state you are casting it to an intger
* Casting an int to a double would be an example of widing, going from smaller byte size to larger byte size.
* Casting double to an into is narrowing, because your are going from larger bytes to smaller bytes, Note, you tend to lose degrees of accuracy.

## Type Checking

* Is Operator
  + Used if your not sure of which type of object you are getting
  + Checks for the type of the given obj
* Typeof operator
  + Returns the object of the type given
  + Gives the CTS version of the type
  + Takes the type of an argument and then returns the marked type

## Discussion

Q: What is Variance? How is it related to polymorphism?

A: Variance is how you implement polymorphism. Covariance Preserve the assignment compatibility.

Q: What are the different type of variances?

A: Covariance, Contravariance, and Invariance

Q: What are the different ways to typecast? (For further reading, Try [this](https://docs.microsoft.com/en-us/dotnet/standard/generics/covariance-and-contravariance))

A: Boxing and unboxing; If you want to convert integers into string you could use the (String) variable.

Q: What are the different ways to check a type?

A:

Q: Why is type checking important?

A: