**Boxing/Unboxing:** Boxing is the process of warping a value type in a System.Object and stores it on managed heap. Unboxing extracts the value.

**Access Modifiers:** Public, private, protected, internal, protected internal

**Protected:** Child class only

**Internal:** Within the same dll

**Sealed:** Cannot be inherited.

**Virtual:** Allows for a property, method, indexer or event to be overridden in a derived class. Must have a method definition. By default, methods are non-virtual.

**Override:** When your subclass is replacing the definition of a parent classes **Virtual** method.

**Abstract:** Indicates there is an incomplete implementation. Deriving classes must complete the definition.

**Constant** is known as “const” keyword in C# which is also known immutable values which are known at compile time and do not change their values at run time like in any function or constructor for the life of application till the application is running.

**Readonly:** Property can only be set in the ctor or declaration of the class.

**Static:** Only a single instance will ever exist at one time. Static members can only be referenced through the type name. Static. Cannot have any instance members.

**Overload:** Different number of parameters in the signature.

**Heap:** Block of memory where objects live.

**Stack:** block of memory for storing local variables and parameters (grows and shrinks, think Stack Trace).

**Ref:** The value is passed by reference. Must be initialized before it is passed. Any change to the argument in the called method is reflected in the calling method.

**Out:** The value is passed by reference. Does not need to be initialized before it is passed. Must be modified in called method.

**In:** The value is passed by reference. Does not allow called method to modify the argument value.

**Passed By Ref/Value:** By default all parameters are passed by value in C#. However, when the type of the param is a reference type you are passing a reference rather than an actual object.

**Reference Type:** Class, String, Object, Interface. Assigning a Reference type variable to another variable copies the reference to the object but not the object itself

**Value Types:** bool, byte, char, decimal, double, enum, float, int, long, sbyte, short, struct, unint, ulong ,ushort

**Dispose:** Release unmanaged resources because the garbage collector does not do it for you. Can be called at any time.

**Finalize:** Release unmanaged resources like files, network connections. Invoked automatically.

**Expressions:** a sequence of one or more operands and zero or more operators that can be evaluated to a single value, object, method, or namespace.

**Query Expressions:** (LINQ)

**Lambda Expressions:**  Inline Methods have no name but can have input params and multiple statements.

**Delegates:** A reference type that represents references to methods with a particular parameter list and return type. Used to pass methods as arguments to other methods. Examples would be events and callback methods. It can also hold an Invocation List ( or list of many methods) that are processed via FIFO.

**Async/Await:** Primarily used for io, network and cpu calls. When an async method is *awaited* the current method will *return* to it’s calling method and thus not block the current thread. When the *awaited* method completes then a thread from the ThreadPool will *return* to awaiting method and continue execution.

**Compile Time Polymorphism or Early Binding:**

In Compile time polymorphism or Early Binding we will use multiple methods with same name but different type of parameter or may be the number or parameter because of this we can perform different-different tasks with same method name in the same class which is also known as Method overloading.

**Hashtable:** Data structure for keeping unique entries. Requires a Collision resolution algorithm.

**Serialization:** Turning and Object into a JSON/String representation

**Deserialization:** Turning a JSON string into an Object

**Stack:** In memory representation of the call hierarchy and local variables.

**Heap:** Memory for Objects

**Is:** Checks an object against a given type. if(sender is Employer e){ // use e }

**As:** Casts an object to a given type. Null if it doesn’t cast.

**Run Time Polymorphism or Late Binding:**

Run time polymorphism also known as late binding, in Run Time polymorphism or Late Binding we can do use same method names with same signatures means same type or same number of parameters but not in same class because compiler doesn’t allowed that at compile time so we can use in derived class that bind at run time when a child class or derived class object will instantiated that’s way we says that Late Binding. For that we have to create my parent class functions as partial and in driver or child class as override functions with override keyword.

**interface v abstract**

* A class can implement any number of interfaces. A subclass can use only one abstract class.
* An abstract class can have non-abstract methods (concrete methods) while in case of interface all the methods have to be abstract.
* An abstract class can declare or use any variables while an interface is not allowed to do so.
* In an abstract class all data member or functions are private by default while in interface all are public, we can’t change them manually.

**== vs equals():** == compares reference, equals() compares content

**Immutable:** The data that comes in is not changed or altered. New data is returned.