Basic Terms and Definitions - Designed for C#, Java, and some AngularJS.

Abstract Class (C# & Java)

An abstract class is a class that contains one or more abstract methods. An Abstract method is a method that is declared, but not defined (contains no block of code).

```
public abstract Animal {
    public void eat(Food food) {
      // do something with food
    }

    public abstract voice makeNoise();
}
```

It is later defined in a sub-class. All abstract methods must be defined in the subclass - otherwise it is impossible to create an instance of that class.

Interface class (C# & Java)

An interface is a collection of abstract methods. A class implements the interface, thereby inheriting the abstract methods of the interface. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

Public vs Private vs Protected vs no modifier (Java)

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

+ : accessible
blank : not accessible

Public vs Private vs Protected vs no modifier (C#)

	Class/ Struct	Derived Class Same Assembly	Derived Class Different Assembly	Subclass (diff pkg) (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+		
internal	+	+			
protected internal	+	+	+	+	
private	+				

+ : accessible
blank : not accessible

Enumeration (C#)

An enumeration is a set of named integer constants. An enumerated type is declared using the enum keyword

Function/Method overloading (C# & Java)

Having multiple definitions for the same function name in the same scope. The definitions must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload functions that differ only by return type.

```
class Calculator {
    public int Add(int i, int j) {
        return i + j;
    }
    public int Add(int i, int j, int k) {
        return i + j + k;
    }
}
```

Method Overriding
Two methods with the same name and the same signature [parameters]. One method will be in the base class and the other will be in the derived (child) class, where the base class is inherited by the derived class.

C#:
```
class BaseClass {
    public virtual int Add(int num1, int num2) {
            return (num1 + num2);
    }
}
class ChildClass: BaseClass {
    public override int Add(int num1, int num2) {
        if (num1 <= 0 || num2 <= 0) {
            num1 = Convert.ToInt32(Console.ReadLine());
            num2 = Convert.ToInt32(Console.ReadLine());
        }
    return (num1 + num2);
}
```

Java (no override modifier):
```
class BaseClass {
    public virtual int Add(int num1, int num2) {
            return (num1 + num2);
    }
}
class ChildClass: BaseClass {
    public int Add(int num1, int num2) {
        if (num1 <= 0 || num2 <= 0) {
            num1 = Convert.ToInt32(Console.ReadLine());
            num2 = Convert.ToInt32(Console.ReadLine());
        }
    return (num1 + num2);
}
```

Polymorphism **(C# & Java)**
Polymorphism is often expressed as 'one interface, multiple functions'. Polymorphism can be static of dynamic.

- **Static (or Early Binding)** - Linking a function with an object at compile time. This can be done in two ways, *Function Overloading*, or *Operator Overloading*.

- **Dynamic (or Late Binding)** - The decision of which function to execute is determined at run time. This can be done with *Method Overriding*.

Static Method **(C# & Java)**
A Static Method must be in a Static Class. When accessing a Static Method, you do not need to create an instance of the Static Class, but can. *Java Code Conventions* says you should avoid accessing a Static Method from an Object.

```
public SomeMethod(int personId) {
    //Calling a Static Function
    String personName = PersonRepo.GetPersonNameByID_StaticMethod(personId);

    //Calling a Static Function from instance
    //This may result in a warning
    PersonRepo person = new PersonRepo();
    person.GetPersonNameByID_StaticFunction(person);
}
```

Static Class **(C# & Java)**
A Static Class is a class which can have both instance and static members, and both instance and static methods. A static class can only access static members and methods in the outer class. You can create an instance of a Static Class.

Dependency Injection **(AngularJS, C#, & Java)**
When a class requires (or depends) on an object, a Dependency Injection is when it is injected or passed into the class instead of being created in the class.

This is not a dependency injection:

```
public SomeClass() {
    myObject = Factory.getObject();
}
```

This is a Dependency Injection:

```
public SomeClass(MyClass myObject) {
    this.myObject = myObject;
}
```

Action Method **(Java)**

An Action Method is the first method in a controller that a users request will hit. In a controller, there are can be many Action Methods. Action Methods usually have one-to-one mapping with user interactions. An user interaction can be a specific URL was entered, a form was submitted, or a link was clicked.

Action Filter **(C#)**

Action filters are used to implement logic that gets executed before and after a controller action executes. Action Filters are additional attributes that can be applied to either a controller section or the entire controller to modify the way in which an action is executed.

- **Output Cache** − This action filter caches the output of a controller action for a specified amount of time.
- **Handle Error** − This action filter handles errors raised when a controller action executes.
- **Authorize** − This action filter enables you to restrict access to a particular user or role.

Action Result **(C# & Java)**

Each action needs to return an Action Result. Many cases the client sends a request to the server, the Action Method picks up on that request, does some stuff, and decides what view to send the client to. That view is returned in the action result. There are many different types of Action Results.

- **ViewResult** - if you want to return a View
- **FileResult** - if you want to download a file
- **JsonResult** - if you want to serialize some model into JSON
- **ContentResult** - if you want to return plain text
- **RedirectResult** - if you want to redirect to some other action
- **HttpUnauthorizedResult** - if you want to indicate that the user is not authorized to access this action
- **FooBarResult** - a custom action result that you wrote

Two-Way Binding **(AngularJS)**

Data binding in AngularJS is the synchronization between the model and the view.
Two-way binding just means that:

1. When properties in the model get updated, so does the UI.
2. When UI elements get updated, the changes get propagated back to the model.

Power Shell **(Microsoft)**

PowerShell is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language. Initially a Windows component only, known as Windows PowerShell, it was made open-source and cross-platform on 18 August 2016 with the introduction of PowerShell Core.[4] The former is built on .NET Framework while the latter on .NET Core.

Singleton Class **(C# & Java)**
A Singleton class is a class that can have only one object at a time. After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. In a normal class, we use a Constructor() method, while in a Singleton class, we use getInstance().

```
class Singleton {
    private static Singleton single_instance = null;
    public String s;

    private Singleton() {
        s = "Hello I am a string part of Singleton class";
    }

    public static Singleton getInstance() {
        if (single_instance == null)
            single_instance = new Singleton();

        return single_instance;
    }
}

class Main {
    public static void main(String args[]) {
        Singleton x = Singleton.getInstance();
        Singleton y = Singleton.getInstance();
        Singleton z = Singleton.getInstance();

        x.s = (x.s).toUpperCase();

        System.out.println("String from x is " + x.s);
        System.out.println("String from y is " + y.s);
        System.out.println("\n");

        z.s = (z.s).toLowerCase();

        System.out.println("String from x is " + x.s);
        System.out.println("String from y is " + y.s);
    }
}
```

Output
```
String from x is HELLO I AM A STRING PART OF SINGLETON CLASS
String from y is HELLO I AM A STRING PART OF SINGLETON CLASS

String from x is hello i am a string part of singleton class
String from y is hello i am a string part of singleton class
```

Linq Lazy Loading (C#)
Lazy Loading is when a child object is not loaded automatically with its parent object until they are requested. Lazy Loading is the default in Linq and in SQL. If you try to access multiple related objects with lazy loading off, then Linq will call the database multiple times. For example,

```
var query = context.Categories.Take(3);
foreach(var Category in query) {
      Console.WriteLine(Category.Name);

      foreach(var Product in Category.Products) {
            Console.WriteLine(Product.ProductID);
      }
}
```

This will create four separate queries and call the database once. It may be better to disable lazy loading for this query. If you did, then the database would be called only one time.

SQL Stuff

- Primary Key
- Foreign Key