# Asynchronous and Parallel Programming

**Asynchronous:**Hey you, do this task and return with the results. Meanwhile I will get done other tasks. (No wasting of time. :))  
  
**Parallel:** Hey you, do this task and return with the results. Meanwhile I will do nothing but wait for you. You can smartly use other free resources/cores of the CPUs/machines to boost the work time.  
  
Parallel programming is like developers (a team) working on a project module with a common goal to complete the task (project module job work). :)

break your task into independent pieces of work, then what? Would you not like to get these independent pieces of work done in parallel to finish faster? This is called parallelism. In this case there must be available free resources to do the job in parallel.

# How to identify extension method?

Here **this** key word is used with argument

public static int WordCount(this String str) {

return str.Split(new char[] { ' ', '.', ',' }).Length;

}

# What is velocity in Scrum?

Velocity is a measure of the amount of work a Team can tackle during a single Sprint and is the key metric in Scrum. Velocity is calculated at the end of the Sprint by totalling the Points for all fully completed User Stories. Estimated time for this course: 5 minutes.

# Impediments in Scrum?

An Impediment is anything that keeps the Team from getting work Done and that slows Velocity. Impediments come in many forms: a sick team member, a missing resource, lack of management support or even a cold team room. If it's blocking the team from doing its work, it's an Impediment.

# Retrospective in Scrum?

# Scrum is a process of continuous improvement, and a retrospective Scrum is a time for teams to reflect on the opportunities to accomplish this. The sprint retrospective is a recurring meeting dedicated to discussing what went well and what can be improved in a sprint.

# https://dotnettutorials.net/lesson/composite-design-pattern/

# Composite design pattern?

# The Composite Design Pattern allows us to compose one whole object from one or more individual objects to represent a part-whole hierarchy.

# In simple words, we can say that the Composite Design Pattern composes the objects in the form of a tree structure to represent part as well as a whole hierarchy. This design pattern is used when we need to treat a group of objects in a similar way as a single object

# A composite object is an object which contains other objects. The point that you need to remember is a composite component may also contain other composite objects. The object which does not contain any other objects is simply treated as a leaf object.

# 

# Creating IComponent Interface

# namespace CompositeDesignPattern

# {

# // The base Component class declares the common operations for both simple and complex objects.

# public interface IComponent

# {

# void DisplayPrice();

# }

# }

##### Creating Leaf Class

# using System;

# namespace CompositeDesignPattern

# {

# // The Leaf class represents the end objects.

# // A leaf can't have any children.

# // The Leaf object is the Object which does the actual work

# public class Leaf : IComponent

# {

# public int Price { get; set; }

# public string Name { get; set; }

# public Leaf(string name, int price)

# {

# this.Price = price;

# this.Name = name;

# }

# public void DisplayPrice()

# {

# Console.WriteLine($"\tComponent Name: {Name} and Price: {Price}");

# }

# }

# }

##### Creating Composite Class

# using System.Collections.Generic;

# namespace CompositeDesignPattern

# {

# // The Composite class represents the complex components that have children.

# // The Composite objects delegate the actual work to their children and then combine the result.

# public class Composite : IComponent

# {

# public string Name { get; set; }

# //The Object is used to hold all the child components of a composite components

# List<IComponent> components = new List<IComponent>();

# //The Constructor takes the Composite name as the input parameter

# public Composite(string name)

# {

# this.Name = name;

# }

# //The following Method is used to add Child Components inside the Composite Component

# public void AddComponent(IComponent component)

# {

# components.Add(component);

# }

# //Display the Price of Composite Components

# public void DisplayPrice()

# {

# foreach (var item in components)

# {

# //Delegates the work to the actual leaf object or child components

# item.DisplayPrice();

# }

# }

# }

# }

# Client Program

# using System;

# namespace CompositeDesignPattern

# {

# public class Program

# {

# static void Main(string[] args)

# {

# // The client code works with all of the components (Both Leaf and Composite) via the base interface i.e. IComponent.

# // IComponent means the class that implements the IComponent Interface

# //Creating Leaf Objects or you can say child objects

# IComponent hardDisk = new Leaf("Hard Disk", 2000);

# IComponent ram = new Leaf("RAM", 3000);

# IComponent cpu = new Leaf("CPU", 2000);

# IComponent mouse = new Leaf("Mouse", 2000);

# IComponent keyboard = new Leaf("Keyboard", 2000);

# //Creating Composite Objects

# Composite motherBoard = new Composite("MotherBoard");

# Composite cabinet = new Composite("Cabinet");

# Composite peripherals = new Composite("Peripherals");

# Composite computer = new Composite("Computer");

# //Creating Tree Structure i.e. Adding Child Components inside the Composite Component

# //Adding CPU and RAM in Mother Board

# motherBoard.AddComponent(cpu);

# motherBoard.AddComponent(ram);

# //Adding Mother Board and Hard Disk in Cabinet

# cabinet.AddComponent(motherBoard);

# cabinet.AddComponent(hardDisk);

# //Adding Mouse and Keyboard in peripherals

# peripherals.AddComponent(mouse);

# peripherals.AddComponent(keyboard);

# //Adding Cabinet and Peripherals in Computer

# computer.AddComponent(cabinet);

# computer.AddComponent(peripherals);

# //To Display the Price of the Computer i.e. it will display the Price of all components

# Console.WriteLine("Price of Computer Composite Components");

# computer.DisplayPrice();

# //To display the Price of the Keyboard

# Console.WriteLine("\nPrice of Keyboard Child or Leaf Component:");

# keyboard.DisplayPrice();

# //To display the Price of the Cabinet

# Console.WriteLine("\nPrice of Cabinet Composite Component:");

# cabinet.DisplayPrice();

# Console.Read();

# }

# }

# }

# What is Repository?

# A repository is nothing but a class defined for an entity, with all the possible database operations. For example, a repository for an employee entity will have the basic CRUD operations and any other possible operations related to the Employee entity.

# Most data-driven applications need to access the data residing in one or more other data sources. Most of the time data sources will be a database. Again, these data-driven applications need to have a good and secure strategy for data access to perform the CRUD operations against the underlying database. One of the most important aspects of this strategy is the separation between the actual database, queries, and other data access logic from the rest of the application. In our example, we need to separate the data access logic from the Employee Controller. The Repository Design Pattern is one of the most popular design patterns to achieve such separation between the actual database, queries, and other data access logic from the rest of the application.

# Which Loggin is supported in .Net Core?

# NLog, Serilog, Logger, Log4Net, elmah and others

# Write logger in file using NLog?

# https://www.youtube.com/watch?v=o5u4fE0t79k

# Download NLog.Web.AspNetCore Package for Core latest version from nugget

# Add txt file in Project and name it as nLog.Config

# Above file contains target file name date wise

# It also contain logging rule

# 

# To Copy NLog file to output directory, right click on NLog.Config file and set

# Copy To Output directory=” Copy if Newer”

# With other setting add code like Logging.AddNLog in Program.cs in

# CreatWebHostbuilder method

# 

# Different Log Level?

# 

# 

# What is GIT & GITHUB?

# GIT: is Version control, it is on developer machine. It creates version no’s

# GITHUB: is source control, it is on server machine

# What is Git Clone?

# It’s used to map repository with local folder