C# Advanced

# Reference Types

Classes

Array

Strings

Functions

# Garbage Collector

# Generics

* where T : IComparable
* where T : Product
* where T : struct
* where T : class
* where T : new()

## Generic List

public class GenericList<T> {

public void Add(T value) {

}

public T this[int index] {

get { throw new **NotImplementedException**(); }

}

}

## Generic Dictionary

public class GenericDictionary<TKey, TValue> {

public void Add(TKey key, TValue value) {

}

}

## Constraint to an interface

static T Max<T>(T a, T b) where T : IComparable {

return a.CompareTo(b) > 0 ? a : b;

}

## Constraint to a class

static float CalculateDiscount<TProduct>(TProduct product) where TProduct : Product {

//product.Price;

return 0;

}

## Constraint to a value type

struct Nullable<T> where T : struct {

private object \_value;

public Nullable(T value) : this() {

\_value = value;

}

public bool HasValue {

get { return \_value != null; }

}

public T GetValueOrDefault() {

if (HasValue) return (T)\_value;

return default(T);

}

}

## Constraint to default constructor

static void DoSomething<T>(T value) where T : new() {

var obj = new T();

}

# Delegates

A reference to a function

var filters = new PhotoFilters();

Action<Photo> filterHandler = filters.ApplyBrightness;

filterHandler += RemoveRedEyeFilter;

Process(filterHandler);

// delegate as parameter

public void Process(Action<int> filterHandler) {

filterHandler(photo);

}

// the function to pass as delegate

public void ApplyBrightness(Photo photo) {

Console.WriteLine("Apply brightness");

}

static void RemoveRedEyeFilter(Photo photo) {

Console.WriteLine("Apply RemoveRedEye");

}

# Lambda Expressions

var cheapBooks = books.FindAll(b => b.Price < 10);

# Events and Delegates

public class UserSample {

static void Main() {

NetworkStatus networkStatus = new NetworkStatus();

networkStatus.OnNetworkStatusChanged += NetworkStatus\_OnMultipleOfFiveReached;

networkStatus.CheckNetworkStatus();

}

private static void NetworkStatus\_OnMultipleOfFiveReached(object sender, NetworkStatusEventArgs e) {

Console.WriteLine("Network status is: ", e.IsActive);

}

}

public class NetworkStatus {

public event EventHandler<NetworkStatusEventArgs> OnNetworkStatusChanged;

public void CheckNetworkStatus() {

if (OnNetworkStatusChanged != null) {

OnNetworkStatusChanged(this, new NetworkStatusEventArgs(true));

}

}

}

public class NetworkStatusEventArgs : EventArgs {

public NetworkStatusEventArgs(bool isActive) {

IsActive = isActive;

}

public bool IsActive { get; set; }

}

# Extension Methods

public void Test() {

"sample String".SampleMethod();

}

}

public static class StringExtensions {

public static string SampleMethod(this String str) {

return str;

}

}

# Nullable Types

DateTime? date1 = null;

DateTime date2 = date1 ?? DateTime.Today;

DateTime date3 = (date1 != null) ? date1.GetValueOrDefault() : DateTime.Today;

# Exceptions

public class SampleClass {

public void Sample() {

try {

} catch (CustomException ex) {

} catch (Exception ex) {

} finally {

}

}

}

public class CustomException : Exception {

public CustomException(string message, Exception innerException)

: base(message, innerException) {

}

}

# ASYNC

## Task

public async Task DownloadAsnyc(string url) {

var webClient = new WebClient();

var html = await webClient.DownloadStringTaskAsync(url);

using (var streamwriter = new StreamWriter(@"c:\file.txt")) {

await streamwriter.WriteAsync(html);

}

}

## Task<string>

private async void Button\_Click(object sender, EventArgs args)

{

var getHtmlTask = GetHtmlAsync("http://msdn.microsoft.com");

string str1 = "Waiting for the task to complete";

var html = await getHtmlTask;

string str2 = html.Substring(0, 100);

}

public async Task<string> GetHtmlAsync(string url) {

var webClient = new WebClient();

return await webClient.DownloadStringTaskAsync(url);

}