**Working with null:**

The value is the information

The reference points to the information so reference may point to nothing (null)

primitive datatype can also accept null value provided “?” sign.

int? a

DateTime? dateTime

bool? IsNoob

if (string.IsNullOrEmpty (name))

if (string.IsNullOrWhiteSpace (name))

int days= Player.DaysSinceLastLogin ?? -1;

int days=player ?. DaysSincelastLogin ?? -1;

This first check for player if its null or not and then checks for days and if both are null then only 2nd statement will be assigned.

**Null Conditional operator**

Player [] player=new Player [3]

{

  new Player {Name="Sarah"},

  new Player (),

  null

};

string p1=player? [0]?. Name;

string p2=player? [1]?.Name;

string p3=Player? [3]?.name;

Above all the null condition will be handled even if the whole list is set to null

Null Object Pattern-->Can be discussed.

Method parameter can be validated before processing the request or we can make a separate class for input parameter

Convert.ToDecimal ();                                  //Doesn't handle null value or empty string value

Decimal.TryParse (input, out output);       //should use this

**Exception handling:**

If (string. IsNullOrwhiteSpace (goalSteps)) throw new ArgumentException ("Goal must be entered");

if (! decimal. TryParse (goalSteps, out goalStepCount)) throw new ArgumentException(“”);

**Unit Testing of Exception:**

test [Symbol]tab + tab

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void MethodNameIsNull()

{

  //Arrange

  //Act

  //No Assert here[Symbol]It by default checks for exception

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void MethodNameIsNull ()

{

  //Arrange

  //Act

  Try {

     var actual=methodCall;

  }

  Catch (Exception ex)

 {

     Assert.AreEqual("Exception message thrown in actual message", ex.Meassage);

     throw;

   //here we want to confirm the exception message to same as we are throwing in actual method.

    }

 }

If returning one value is not sufficient and throwing exception is not quite right specially in validation   cases.

In case of Validation we want to return the validation result and also want to store messages out of that processes.

We can use following method four ways to handle this but the last one should be used.

**1. ref keyword**

public bool ValidateEmail (ref string message)

{

  if(...)

  {

    var valid=true;

    message="";

    ...

  }

}

//calling above method

string message=String.Empty;                      //need to initialize the message if we use ref keyword

var isValid= customer. ValidateEmail (ref message);

**2. out keyword**

public bool ValidateEmail (out string message)

{

  if(...)

  {

    var valid=true;

    message="";

    ...

  }

}

//calling above method

string message;                               //don’t need to initialize the message if we use out keyword

var isValid= customer.ValidateEmail (out message);

**3. Tuple keyword**

public Tuple<bool, string> ValidateEmail ()

{

  Tuple<bool, string> result= Tuple. Create (true,"");

  if(...)

  {

    result= Tuple. Create (false, “Email is null");

    ...

  }

}

//calling above method

var result= customer. ValidateEmail ();

**4. using class as return type**

className-->OperationResult

public class OperationResult

{

   public bool Success {get; set;}

   public List<string> MessageList {get; private set;}

   public OperationResult ()

   {

      MessageResult= new List<string> ();

      Success= true;

   }

   public void AddMessage (string message)

  {

      MessageList.Add (message);

   }

}

**Defending Code Constructs:**

If (if (if (.... -->use separate method to execute these nested if block

//use of enum

public enum paymentType

{

  creditCard=0;

  payPal=1;

}

public int PaymentType {get; set;}

switch ((paymentType) this. PaymentType)

{

  case paymentType.creditCard:

      break;

  default:

      throw new ArgumentException ();

}

**Asserts, Errors, Exceptions:**

public OperationResult PlaceOrder (Customer customer, Order order, Payment payment)

{

   if(customer==null) throw new ArgumentNullException ("Customer instance is null");

}

//We can also use global exception handler-->not a good practice

we just throw exception in every method but catch it where these methods are called. And a good practice is to catch specific exception.

Logging of exception: Log only that exception which user can’t fix. And, if user can fix then show just a message to the user.

Sometimes we need to apply Try Catch in a specific method as below. We need email and some relevant information to log the exception which we get here in this method only and we just re-throw the exception using catch block.

public void SendEmail (string emailAddress, string message)

{

  //if valid email address is provided

  try

  {

    send an email.

  }

  Catch (InvalidOperationException ex)

  {

   //log and do if we want to return some custom result. ex can be used in logging.

   throw;     //re-throwing of the exception.

  }

}

if we write "throw ex" then it will throw a new exception in that case exception might lose something such as call stack. So , a good practice is to use only “throw”.