**Inner exception -lec 40:**

**16.** InnerException property returns exception instance that caused current exception.

To retain original exception pass it as a parameter to the constructor of the current exception.

Always **check if inner exception is not null** before accessing any property of inner exception object else will get **Null Reference exception**.

To get the type of inner exception use **GetType** method.

try{

try{

//code for example: file handling

} catch(Exception exception){

// for handling FileNotFound exception.

}

} catch(Exception exception){

exception.GetType().Name // to get parent exception type and

If(exception.InnerException !==null){

exception.InnerException.GetType().Name // to get inn er exception type.

}

}

exception.GetType().Name // to get parent exception type and

exception.InnerException.GetType().Name // to get inner exception type.

**Custom exception – lec 41:**

**17.** We need to create our own exceptions some times.

We can track original exceptions using inner exceptions by overloading our custom exception class constructor accordingly.

Similarly for custom serialization .

[Serializable]

public class UserLoggedInException : Exception{

public UserLoggedInException (): base(){}

public UserLoggedInException( string message): base(message){}

public UserLoggedInException( SerializationInfo, StreamingContext context) : base(info, context){}

}

**18.** Do not abuse try catch exception handler, instead at times you can use error checking mechanisms like tryIntParse().

**19.** ENUMS Lec - 45

* They are strongly typed constants.
* If a program contains **set of integral numbers** use Enums to make it more readable and maintainable.
* Enums's underlying type is int by default.
* Since enums are strongly typed we cannot convert them to integer type implicitly. We need to give explicit type casting.

Gender gen = 3; // wrong

Gender gen = (Gender)3; // correct

* Enums are of value types.
* "enum" keyword is used to create enumerations, whereas "Enum" class has static methods like GetNames() and GetValues() which can be used to list enum underlying type values and Names.

**20.** A console or windows application generates executable( exe) and class or web application will generate dll.

**21.** lec 51:

* All the 5 access modifiers can be used for **type members** but only public and internal can be used for **types** (classes etc).
* Default access modifier for **Type members are private** and for **types are internal**.

**22.** lec 52: Attributes

* Attributes allow you to add declarative information to your programs.
* Which can be queried later at runtime using reflection.
* Some of the pre-defined attributes are – Obsolete, WebMethod and serializable.

Public helloAttribute­­ : Attribute{}

**23.** lec 53: Reflections:

* Reflection is the ability of inspecting an assemblies' metadata at runtime. (After compilation , assembly contains 2 parts- metadata and intermediate language).
* Used to find all types in an assembly and/or dynamically invoke methods in an assembly.

USES:

1. Used by IDE or UI designers in winforms etc.
2. Late binding – Dynamically creating instance of a type , about which we don’t have info during compile time.
3. To access alternate implementations of an interface for example - we allow user to pick one or the other using a config file by reading name of the class using reflection from the config file.

* **Type** class is important when we deal with reflections.

Type T = Type.GetType("Preetham.Customer");

Type T = typeOf(Customer);

Type T = Customer.GetType();