CASE – TRYCATCHMECHANISMS

Dynamic Annotations Example

//enable dynamic: type checking done at runtime

//

package trycatch

namespace trycatch {

@dynamic

import case.lang.System

import case.lang.System.IO

import case.lang.IOStream

import case.lang..PrintStream

import case.lang.System.IO.IOException

import case.lang.System.ArrayIndexOutOfBounds

import case.lang.System.IO.PrintWriter

import.case.lang.Processor.ProcessAsync

import case.lang.Processor.ExceptionLogger

import case.lang.Processor.HandleExecptionAsync

import.case.lang.FileInputStream]

Try Catch(IOException io) { Processor processor, await processor.ProcessAsync)}

{ ExceptionLogger logger = new ExceptionLogger(),

await logger.HandleExceptionAsync(exception);} code {\} }

String->Object->Main

#public class Program

[public Program(String [] args)

[EntityPool Pool = EntityPool.getEntityPool]

assert(Pool) //asserts that Pool exists and has a value

Stream (n) String

dynamic TempVariable (n) String

//turn on dynamic typing for TempVariable of type String

Stream (k) Int

dynamic TempVariable (k) Int //change dynamic type

//Try Catch Finally mechanisms

//Try Catch Finally – useful but tends to obfuscate code

Stream (l) PrintWriter

Out (l) PrintWriter

Try Catch(IOException PrintWriterException) { out = PrintWriter } //

{ Out.close() } code {\}

**Await in Try Catch**

The await keyword can be called inside the catch and finally blocks. This opens up the way to perform an async exception handling or fallback process in case an exception happened during an async process call.

[public async void Process()

Try Catch(IOException io) { Processor processor, await processor.ProcessAsync)}

{ ExceptionLogger logger = new ExceptionLogger(),

await logger.HandleExceptionAsync(exception);} code {\} }

]

**AutoFall**

An improvement on try….catch statements. Classes and or operations that are normally required to have to declare a try statement blocks like IOStream exceptions and FileStream etc. all must, if they use this new method, implement autofall. For example:

Old method:

Try

{  
 FileInputStream fis = new FileInputStream();

fis.doSomething();

} Catch (Exception eo) {

PrintStream;

} Finally {

Error():

}

With AutoFall you know at one point the current thread will terminate it because it finishes a data stream from the file source or keybaord input, etc. So we will let the compiler and its magic handle try….catch, etc. As such,

**AutoFall** FileInputStream Fis;

Fis.doSomething();

It you can even chain them toegether:

**AutoFall** FileInputStream fis;

Fis.doSomething();

**AutoFall** IOStream ios;

Ios.doSomething():

In the above, it executes both in parallel and both only alert you if you there is an exepction.

#end class

}