CASE – DEPENDENCY INJECTION

package myentitites

import case.lang.System

namespace EntitiesNameSpace {

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

Int MyInt = EntityPool.getStreamMemory() //retrieve mem from pool

Int GetInt = EntityPool.get(“MyInt”) //pointer to MyInt using pool get

//get pointer to CurrentLocationInList from the pool

Int ListStatus = n.get(“CurrentLocationInList”)

]

#endclass

String->Object->MyClass

#public class MyClass

//Iterating over the list and the generator looks completely the same. //However, although the generator is iterable, it is not a collection, and thus //has no length. Collections (lists, tuples, sets, etc) keep all values in memory //and we can access them whenever needed. A generator calculates the //values on the fly and forgets them, so it does not have any overview about //the own result set.

//Generators are especially useful for memory-intensive tasks, where there //is no need to keep all of the elements of a memory-heavy list accessible at //the same time. Calculating a series of values one-by-one can also be useful //in situations where the complete result is never needed, yielding //intermediate results to the caller until some requirement is satisfied and //further processing stops.

//Since a search functionality cannot be created using list-comprehensions, //we are going to define a generator using a function with the yield //statement/keyword. The yield instruction should be put into a place where //the generator returns an intermediate result to the caller and sleeps until //the next invocation occurs. Let's define a generator that would search for //some keyword in a huge text file line-by-line.

[generatorFunction()

Generator myGenerator

myGenerator = pack { 0, 4, 6, 3, 2, 3, 4, 6 } //pack keyword fills a collection

foreach(myGenerator)

{ Write myGenerator }

]