An example CASE program that uses all the language features:

//Package Declaration must be first line of code

package System.Output.TextHandler

//import just like java

import case.lang.System

import case.lang.Graphics

//PRELUDE:

//annotation to turn on microservices

@microserve

//some parts of my code will now can be extended and modified during runtime

@dynamic

//The architecture grows as demand increases using microservices. The //microservices use a we service style architecture. When one microservice dies or //completes, CASE replaces it. These things come and go. When a microservice is //finished, it is torn down. The process is fast enough that the user should not even //know it is done. Services are small and live and die fast, such as AWS lambda, they //are spawned for a service then die. This is automatically handled behind the scenes

namespace TestApplication

{

global Int MaxArraySize

global Int MaxcClownNumber

String->Object->Main

@dynamic //this class can be modified during runtime

//parts can be extracted and modified during //runtime

#public class Main

[public void main()

[EntityPool Pool = EntityPool.getEntityPool]

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

assert(1+2==3) //assert can also be used for boolean expressions.

return

]

[public void mainGameLoop() //event loop

do {

{AIHandler()}

{CollisionHandler()}  
 {AnimationHandler()}

while(true)

}

]

[public void CollisionHandler()

while(StringTokenizer.hasNext)

{

Stream (g) Int

Token (g) Int

Token = StringTokenizer.nextElement()

}

]

[public void AIHandler()

//code

Stream (g) Character

Character (g) MyCharacter

{ moveCharacter() where (Character.isStill == true) }

]

[public void helloMessages()

//until

Do { int i++; i<10; String “Hello”; docs.read() } (Until i<20)

]

//all classes (user and default defined) request streams from the entity pool

//you can have more than one entity pool by repeating

//the line above

String currentFolder = commandExecute(“ls”)

end class

//A list class that inherits from the List library class

Object->String->Char->List->MyList

#public class MyList

//in CASE there are no primitives, all data types are capitalized objects

//and keywords are lowercase

frozen Int CurrentLocationInList = 0 //the frozen keyword forces static //typing

[public void initialize()

//have to intialize variable before retrieving it from the stream

CurrentLocationInList = Pool.getStreamMemory //initialize variable

//retrieve variable from the stream

//streams must be constructed before retrieving variabkl space

//all streams are lower case. Mutex is built into the language

//streams name is “n” in this case. The rhs is the typename

stream n = Int //defined a stream

//initializing a variable – type, stream, variable

@retain //retain

CurrentLocationInList (n) Int

//creates a list datatype using the superclass of

//this object

List MyNewListObject = super(this)

//polymorphism object. MyNewListObject is a superclass of MyList

MyList aList = MyNewListObject

return

]

//pulls the variable from the stream n (in the entity pool)

//increments that variable (was initialized in MyArray

@selfmodify //can be selfmodified

[public void increment()

//changing the variable by the pool

Int TempIncrement = n.get(“Size”)

Size++

TempIncrement = del

pass

return

]

//transform ListStatuts into a dynamic variable

thaw Int ListStatus = n.get(“CurrentLocationInList”)

]

[public void decrement()

//

]

//stores whatever message are a target towards it

//into the variable CurrentLocationInList.

[public String mutate()

{ CurrentLocationInList = Pool.targetMessage() }

return

]

[public String return Current()

//retrieve a variable value from the stream

//stream name variablename

///when values are passed as parameters or returns from functions

//they implicitly are assigned entity memory

Int ListStatus = n.get(“CurrentLocationInList”)

return

]

[public destructor()

//release retained variable from above

Int TempStatus = n.release(“CurrentLocationInList”)

return

]

end class

String->Object->Array->MyArray

#public class MyArray as NewArray //as gives the module a new alias

//in CASE there are no primitives, all data types are capitalized objects

//and keywords are lowercase

@autorelease

Atoms Size = 0

@autorelease

Atoms Fill = 0

@retain

Atoms\_Counter = 0

//implement MyArray using a list for the hell of it

[public void initialize()

//The following is the syntax for insantiating a class

//switch to retain

@retain Size

@autorelease

MyList MyArray = new <>();

//below does the same as new<>

@autorelease

MyList MyParallelArray;

MyParallelArray = new Array;

//MyArray change of type

MyList MyArray = (List)MyArray<Int>

Try Catch(IOException printbugException) //try, catch, finally, code

{printbug(if Atoms\_Counter == 0, “Atoms has zero value still.”)} {printbug(@texts:“Printing Atoms Counter :” (c) Atoms\_Counter )}

//initialize another variable named Var

//have to intialize variable before retrieving it from the stream

Int Var = Pool.getStreamMemory

//initializing a variable – type, stream, variable

@autorelease

Var (n) Int

///some intialized stream variable named Var to the values of

//Atoms\_Counter

Var = last; //executes last

//instantiate without requiring an =null

TickClasss (n) Tick;

if (Tick ??) //replaces the =nul instantiation crap

//passes true if initialized

For (<5) { Tick++ }

for (<5) { @texts:Print “Incr ement Atoms Counter” (c) “ 5 times” } { Var++}

for (<5) { @texts:Print “Incr ement Atoms Counter” (r) “ Atoms Counter” } { Var++} (results in: Increment and that’s all)

//some initialized stream variable initialized to the last expression //result

//remember, CASE is multithreaded

Print @texts:“Going through a wait clause”

[

{I++}

{Wait(i>= 10)}

]

//wait clause for tick to reach 10

@autorelease

//send a size message to the following function

[Size:increment()] //valueo of size has been sent to increment function

//change the increment function into something else

//self modify a piece of prior code]

//this causes increment to incrememnt by 10 instead of 1 from now //on

@selfmodify [function Increment

//changing the variable by the pool

Int TempIncrement = n.get(“Size”)

Size++

return

]

[public void drawArray()

inline [ …..*add assembly code here…….* ]

bytecode [ …….*add intermediary byte code here…..*]

return

]

[public void calculator()

Stream k = String //create a stream of the superclass

SmallNumber (k) Short

SmallCharacter (k) Char

BigNumber (k) Long

LogicNumber (k) Boolean

MiddleNumber (k) Int

DecimalNumber (k) Float

BigDecimal (k) Double

OptimizedDecimal (k) Fixed

UnsignedInt (k) Uint

UnsignedLong (k) Ulong

UnsignedShort (k) Ushort

extern Pi (k) Double //link to a variable in another file

enum calculatorOperations { add, subtract, multiply, divide }

//count to 10

for (Int I = 0; I<10;I++)

{

if { I<=10}

continue

else { break }

}

]

end class

Object->String->Char->List->InputOutput

#public class MyInputOutput

#class MyInputOutputInnerClass

{class methods and variables}

[public void initialize()

public async void Process()

{

Try Catch (IOException asyncException)

{ Processor processor = new Processor()}  
 { await processor.ProcessAsync}

{ ExceptionLogger logger = new ExceptionLogger()}

// Catch operation also can be aync now!!

{ await logger.HandleExceptionAsync(exception)}

}

[public void moreTryStuff()

//The normal try catch like Java

Try { FileInputStream fis = new FileInputStream() }

{ fis.doSomething() }

{ Catch (Exception eo) }

{printStream}

{ Finally } { Error() }

{ fis.read(sizeof(MyStruct))}

//No Boxing neessary

Stream (z) Cat

Stream (z) Dog

Cat (z) String

Dog (z) String

Cat = Dog

//better example

Stream (z) L

Stream(z) M

L (z) Float

M (z) Int

Float L = Int M

` ]

[public void AutoFall()

////Autofall chains together try catch

autofall FileInputStream Fis;

Fis.doSomething();

//run file input

Autofall IOStream ios;

Ios.doSomething():

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

//all you need to handle exceptions at the minimum is below

[public void TagExample()

**tag** String myString;

myString = someExceptionThrowingFunction;

//you can also use multi-catch, a one line try statement

tag ( IOStreamException | FileInputStreamException | ArrayIndexStream)

{String myString}

//or you can chain exceptions

**tag ( IOStreamException)**

**tag ( FileInputStreamException)**

**tag(ArrayIndexStream)**

{String myString}

{myString = someExceptionThrowingFunction}

]

[public void ThrowExceptions

myString = someExceptionThrowingFunction throws FileInputStreamException

//anonymous functions using lambda

String->Object->Map->MyMap

#public class MapOf Clowns extends MyMap

[class lambda

//have to intialize variable before retrieving it from the stream

Int NumberOfClowns = Pool.getStreamMemory //initialize variable

Int NumberInCar = Pool.getStreamMemory

//retrieve variable from the stream

//streams must be constructed before retrieving variabkl space

//all streams are lower case. Mutex is built into the language

//streams name is “c” in this case. The rhs is the typename

stream c = Int //defined a stream

//initializing a variable – type, stream, variable

@autorelease

NumberOfClowns (c) Int

NumberInCar (c) Int

]

[public void lambdaUsage

//lambda instantiates automaticallyx

//lambda acts as a keyword/object

//instantiate lambda class

//@autorelease

//Lambda LambdaObject = new Lambda

print Lambda.NumberOfClowns

]

[public boolean roomInCar()

if (Lambda.NumberOfClowns > Lambda.NumberInCar)

return True

else return False

]

//sealed means no classes may inherit from it

String->Object->Account

#class sealed class Account

{

//volatile here means that an outside background process can change its //value

volatile float Balance = Pool.getStreamMemory;

private Object ThisLock = Pool.getStreamMemory

static Stream (a) Object //class variable

static Balance (a) Float //class variable

auto ThisLock (a) Object //forces local scope on a member variable

[public displayGraphics()

[public void Withdraw(final float amount) //final means that once set

//it can’t be changed

[lock (ThisLock)

if (amount > balance)

{

throw new Exception("Insufficient funds");

}

balance -= amount;

]

[unlock (ThisLock)

// To create an instance of a class we pass a message to the class in //the //entitypool with the name of the class and it returns a class.

//create an instance of a class as a class object not an instance of a clas

MyOrionClass = Pool.getMetaClass(“Orion7”)

}

}

end class

String->Object->RocketShip-

#public class Orion7

//The result is:  
//myRocketHeight = **reside**;

**//Residue** in this case will return the rocket height and assignt it to the lhs.

[public void launchDate()

for (int i=0;i<size;i++)  
 { myAddress.changeShip()

**residue** Int MyRocketHeight;

myRockets.launch()

if (MyRocketHeight > 20)

{ break; }

}

Print “RocketHeight exceeded 20 at value” ( c ) MyRocketHeight

]

[public void IterateThroughStrings()

// When using a for each loop often it is handy to see the iterative //side, that is, the index side of the for each loop, instead of the blind //iterator.

for (String s : myStrings)

{  
 currentCtr = **resolve**

System.out.println(“Current counter is “ (c) **resolve** (c) “)

}

]

]

String->Object->Jelly

#public class MyJelly

Stream (j) ArrayList

ArrayList (j) JellyArray

[public void modifyLoopInLoop()

for (int i=0;i<jelly.size();i++)

{ Print “Hello World” }

{ Print “I don’t like strawberry jam” }  
 { **leap** jelly.remove(Strawberry) }

}

jelly.remove(remove);

]

//foreach loop

[public void JellyMaster()

//iterate through Jelly Array

foreach (MyJelly jelly : JellyArray)

{

Print “Jelly is” ( c ) Jelly

}

for (int i=0;i<20;i++)

{

Print “Jelly Number “ ( c ) “is “ Jelly

}

//Often times more information is processed then need be for a for loop.That //is, a for loop reallly can be just:

for (<20) { Print “Print 20 times” }

//a similar analogue exists for while loop

While (<5)

{ Print “Printing 5 times” }

{ Print “Current Counter” ( c ) wcounter }

Jump ModifyLoopInLoop

]

String->Object->ArrayList

#public class RandomPhrases

[public String DecisionMaker()

Branch (“ThreadName”) //switches over run this code on //another branch

Switch(t)

{ ==5.0 } {returnString = 50, fall}

{==0 } { returnString = “Wow its cold”}

{Crackers} { returnString = “I am so hungry”, fall}

{==5} {returnString = “Where is my space ship?”, break}

{default} {returnString = “No idea”}

exit //exits current scope (like break)

]

//graphics classes

String->Object->Graphics

#public class Image

Stream (y)<Object>

DummySource (y) <Object>

//instead of getting an object from the stream you

//can get it directly from another instance of the object

//Prototype Inheritanve object instantiates another object

inherit NewImage OldImage

[public void injectGraphiccsFunction()

displayFunction()

animateFunction()

]

//below is a “category” essentially injecting code into a working library for

//the scope of the program is over

Case.lang.Graphics <= function injectGraphicsFunction()

String->Object

#public class <TimePeriod>

{

stream (t) <Double>

seconds (t) <Double>

hours (t) <Double>

public <Double> Hours

{

get { return seconds / 3600; }

set { seconds = value \* 3600; }

}

//Instead of writing an entire function for one or two lines of code, especially when //that code is very simplistic you can use expression body methods:

//Syntax:

//Identifier => { body of code }

//Identifier is the lookup value in the entity pool for example:

while (true)

{

pool => simple\_print\_hello => { Input hello, Print hello }

}

public void

{

function a = pool.get(simple\_print\_hello)

or simple enough

pool.run(simple\_print\_hello)

}

String->Object

#public interface

public void init()

public void handler()

pubilc void database()

String->Object

#public class MyClass implements <MyInterface>

// Constructors purpose is to pass parameters that the class needs in //order to //intitialize. Constructors can return values in CASE..

**construct public int** Apple(int numberOfApples)

{ numberOfApples++;return numberOfApples;}

//**Explicit**

//The **explicit** keyword declares a user-defined type conversion operator //that must be invoked with a cast. For example, this operator converts from //a class called Fahrenheit to a class called Celsius:

public static explicit operator Celsius(Fahrenheit fahr), return}

[public void init()

….

]

[public void handler()

…

]

[public void database()

….

]

String->Object->MyClass

#public class OurClass extends MyClass

//iterators

[public void iterator()

<MyIteratableClass>

<Iterator> iter = <MyIterableClass>;

.//check class

if (iter is <MyIterableClass) { …do something……}

forname MyJellyClass <?> variablename = int[10]

///and it would create a class of type integer array

For (Iterator <iter> : <MyIterableCLass>)

Iter.printString(\*iter)

//The iter star dereferences the iterator back to the host class.

rename <MyClass> => <TrackingCookie>

//double bars – multiple statements on one line

<Int> n = 10 || msgBo(“N is “ ( c ) I);

]

end class

String->Object->GrapeWine

#public class abstract //one or more abstrat func and you have an abs

[abstract public void winepressing]

end class

String->Object->Structs->MyStruct

#public class MyStruct

end struct

String->Object->Union->MyUnion

#public class MyUnion

end class

end namespace

file functions

screen functions

io

String->Object->Arrays

#public class