**COMP 3002 Winter 2025 Assignment #1a**

**Getting Used to Swift**

First lecture Mon Jan 6. 6pm, Assignment due date Sun Midnight Jan 12.

Build a notepad file containing all the code you tested and the execution results.   
Hand-in the code for the last question in a separate notepad file.

Put everything in a folder with your name and ID on it.   
Ignore (by correcting) little errors when you find them.

**Starting Up**

0. Assuming you are running on a Mac machine, the first thing to do is create a folder for course 3002 where you can place code for all the assignment. Create a project Assignment1 and add a file, perhaps called “main” (the name is not important) into which you can place code that you will write and another file called “extensions” into which you can place all your extension code. Alternatively, setup a virtual machine for a Mac if you are on a Windows machine. I also tried “Swift for Windows”; the problem with it is that it’s an older version of Swift and it’s missing some facilities that you need, namely to be able to create your own objects that are hashable. It indicates that a newer version is about to be released (but that message is at least a couple years old). There seems to be other alternatives around but you have to buy them…

**Dealing with Arrays containing many types of objects**

1. Try creating a variable that contains a bunch of different types of objects, like

nil true false "hi" "h" 10 10.2 [10, 20]

Explicitly type your variable so that the compiler allows you to initialize it with a collection containing the above. If you’re having trouble, try using a let instead so the compiler infers the type. Then loop over all the items in the collection and print them out.

Use the debugger to single step as you loop. That way, you will be able to look at the contents of your loop variable to see what’s in it before you try to print it.

To make it a little harder, do it again printing “type: value”. Most probably, you will have to test each possibility using the type testing keyword “is”. Can you distinguish between a character and a string; e.g., can "h" be both. If you find a nicer way to do it without type testing, let us all know.

**Distinguishing equality from identity**

2. In Smalltalk, you compare a and b for equality using a = b and a ~= b. What is it in Swift? Additionally, what are the operators for identity instead of equality.

Now create 4 Bool variables that can be set to the result of comparing two integers 10 and 20 versus 10 and 10 for equality (inequality), identity (and non identity). What does the compiler complain about.

Apparently, primitive types including arrays and dictionaries are structs (rather than instances of classes). You are not allowed to ask if two such structs are identical because they can’t be. I would prefer that you get the answer false rather than telling me I’m not allowed to ask.

How generally can you compare things for equality? For example, can you change the following to make it work…

Let testEquality1 = [1,2,[3,3]] “compare operator for equality” [1,2,[3,3]]

This tells you something about how general this operator is.

To make use of the identity operator, you will have to create a very simple class, say a Thing class with one instance variable called name (a string).

Now you should be able to create 2 instances a and b and compare them for identity. Additionally, you should be able to execute

let c = a

and compare c with a. So everything works fine for class instances. Or does it. If you create a second class Thingy, can you compare a Thing and a Thingy with identity. So much for claiming to be a general programming language.

The general conclusion to draw from all this is to be careful what you compare!

Dealing with Strings

1. Since strings deal with Unicode characters, it’s not implemented traditionally (see notes). You might have been disappointed when you saw that that you couldn’t say something like

"hello" [0]

However, it’s easy to construct an array of characters; var array = Array ["Hello"]. You can easily access the elements one by one.

**Lambdas (Swift calls them closures; Smalltalk calls them blocks)**

3. In the notes, I’ve provided an implementation of do, just to illustrate how to implement a method that takes a closure and how to use it…

Now, let’s consider implementing Smalltalk’s collect: and select: methods… as an alternative to map and filter. Here’s the Smalltalk code

collect: aBlock

"For each element in the receiver, evaluate aBlock with that element as the argument.   
Answer a new collection containing the results as its elements from the aBlock evaluations."

| answer |

answer := self species new: self size.

self do: [:element |

answer add: (aBlock value: element)].

^answer

Note: the species for an array or an interval such as “1 to: 10” are both arrays. In your code, assume everything is an array.

Test it on the Array [1,2,3,4,5] where the intent is to square each element. While you’re at it, extend integers so that you can say “20.squared ()”.

Here’s the code for select:.

select: aBlock

"For each element in the receiver, evaluate aBlock with that element as the argument.

Answer a new collection containing those elements of the receiver for which aBlock evaluates to true."

| answer |

answer := self species new.

self do: [:element |

(aBlock value: element) ifTrue: [answer add: element]].

^answer

Test it on the Array [1,2,3,4,5] where the intent is to select only the odd elements. While you’re at it, extend integers so that you can say “20.odd ()” and get back true or false (in this case, false).

Now that you implemented collect and select, try out the built-in equivalents filer and map to Swift. xCode for help with these methods to see how Swift describes what they do and to help you decide which is which. Perform a test to replace your use of collect and select by the corresponding map and filter. Do you have to change the code at all?

**Implementing your own version of partitionUsing**

4. PartitionUsing is needed for implementing relations which is why I need you to implement it. Here’s the implementation in Smalltalk… I’ve documented it a little bit more that the actual version in Smalltalk so you can gain an appreciation for it.

partitionsUsing: aBlock

"Example which is half Smalltalk (the code), half Swift (the result)

#(1 2 3 4 5 6 7 8 9) partitionsUsing: [:each | each odd]   
=> [true: [1,3,5,7,9], false: [2,4,6,8]]

#("Hi" "all" "guys" "and" "gals") partitionsUsing: [:each | each [0]]   
=> ["H": ["Hi"], "a": ["all", "and"], "g": "guys", "gals"]]

#(10.5 "all" nil (1 2) "any") partitionsUsing: [:each | each class]   
=> [Float: ["10.5], Undefined: [nil], String: ["all", "any"]] //This won’t work in Swift

Returns a dictionary whose key you computed from the instance (any number of keys) and whose value is a collection of all the values that computed that key.

"

| partitions partitionKey |

partitions := Dictionary new.

self do: [:object |

partitionKey := aBlock value: object. "Runs the block and gets a value back treating it as a key. If the key has nothing associated with it in partitions, associate an empty collection with it. Then get the collection associated with it and add object to it."

(partitions

at: partitionKey

ifAbsent: [partitions at: partitionKey put: OrderedCollection new])

add: object].

^partitions If the key has nothing associated with it, associate an empty collection with it."

In words it says. 1. Ask the block to compute a key for the object in self. 2 If there is already a collection with that key, use it. 3. If there isn’t, create a new empty collection (in Swift, use an array). 4. In either case, add the object to that collection.

Make sure you test it. If it’s not working, be wary of dictionaries and arrays since they are structs. This means that if you ask for one that exists, you will most likely get a copy. It’s not the copy that needs to be modified but the original.

Adding a few extensions to array

5. Consider the list in the notes

appendIfAbsent (anObject) Doesn't append if there using ==.

appendIfAbsent (contentsOf: aCollection) Like above.

appendIfIdenticalAbsent (anObject) Doesn't append if there using ===.

appendIfIdenticalAbsent (contentsOf: aCollection) Like above.

appendIfAbsent (anObject, ifAddedClosure) Closure executes if added

Also, add the equivalent to the following Smalltalk method but call it setEqual. It’s useful for comparing arrays where the order of the elements are arbitrary; e.g. [1,2,3] and [1,3,2] are not = as arrays but they are setEqual.

elementsEqual: aCollection

"Answer true if the receiver and <aCollection> has   
equal elements irrespective of order"

self size = aCollection size ifFalse: [^false].

^(self includesAll: element)

**Final Task**

5. Create a class Truck with the following example class method (that’s what static means) that will run; i.e., for which you can run the code in the first comment. Anything this needs to make it work, you will have to add.

static example1 ()

//Truck.example1

var aTruck: Truck = Truck ()

aTruck.driver ("Jim")

aTruck.addPassenger ("Tom")

aTruck.addPassenger ("Frank")

aTruck.addLoad ("Wheelbarrow")

aTruck.addLoad ("Ladder")

aTruck.addLoad ("Cement")

aTruck.driverDo { (\_ driver: String) -> Void in

print ("\nThe driver is \(driver)")

}

aTruck.passengersDo { (\_ passengers: String) -> Void in

print ("\nOne passenger is \(passenger)")]

}

aTruck.loadDo { (\_ load: String) -> Void in

print ("\nThe back of the truck contains a \(load)")

}

//Method do sequences over everything in the truck. The compiler   
 //won’t allow you to compile the following until func `do` is defined (even  
 //if it contains no code.

aTruck do { (\_ anObject: String) -> Void in

print ("\nIn the truck, there is a \(anObject)")

}