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

**Building the class Relation**

**Due: Sun Midnight Jan 12.**

**Basic goals**: To implement the class Relation described in the notes. A start at implementing the parameterized type Relation is provided below (to make sure we create this class in a relatively standardized way). Be advised that copying code from a Word document into Swift is likely to need the document to be reformatted. Example methods are provided to indicate what we expect you to be able to do by the time it’s finished.

**Be simple if possible…**

Even though the class is known to contain triples, it’s not obvious how to store the triples. The newest version of Swift will allow you to store the data in a Set of tuples. Older versions will not because tuples were not hashable last year. In that situation, you might store a tuple as a 3-element array but you will have to ensure that new entries are not added if they are already there (sets do that automatically).

One nice way to work is to create example class methods (in Swift, static methods) that can be used to show that the class works. For example, run Relation.example1 () or Relation.example2 () below. If it doesn’t work yet, add more methods that make it work. Provide other example methods for showing off other aspects of your class.

class **Utilities** {

static func className(of element: Any?) -> String {

//Usage: Utilities.className(of element)

if let element = element {

return String(describing: type(of: element))

} else { return "nil" }

}

}

protocol **Relatable**: Hashable, Comparable, CustomStringConvertible {

var terseDescription: String { get }

}

extension **Relatable** {

var terseDescription: String { return description }

}

class **Relation**<**Item**: Relatable, **Relationship**: Relatable> {

var triples: Set<(Item, Relationship, Item)> //Set of tuples

//Initializers for constructors.

**init**() {//Empty

//USAGE: let relation1 = Relation<Int,String> ()

self.triples = Set()

}

**init**(from triples: [(Item, Relationship, Item)]) {//Array of tuples

//USAGE: let relation2 = Relation<Int,String> (from: [(10, "<", 20), (10, "<", 30)])

self.triples = Set (triples)

}

**init**(from triples: Set <(Item,Relationship,Item)>) {//Set of tuples

//Usage: let relation3 = Relation<Int,String> (from: relation2.triples)

self.triples = triples

}

var **description**: String {

//Output format: Relation(from [(a1 b1 c1) (a2 b2 c2) …]).

let triplesDescription = triples.map {

"(\($0.0.terseDescription) \($0.1.terseDescription) \($0.2.terseDescription))" }

return "Relation(from: [\(triplesDescription.joined(separator: ", "))])"

}

//The 'from(\_, relationsDo: closure)' method partitions the triples that start with froms and iterates over them.

func **from**(\_ froms: [Item], **relationsDo**: closure (Relationship, Relation) -> Void) {

//For you to fill in the code. Note that froms is NOT a keyword but relationsDo: is a keyword.

}

static func **example1** () -> Void {

//Relation.example1 ()

//First, build a relation.

var relation<Int,String> = Relation (from: [(2 "<" 3) (1 "=" 1)

(3 ">" 1) (2 "<" 4)(1 "<" 5) (5 "<" 6) (2 "<" 5)]).

//Second, show that the entire relation can be printed…

print ("\nLet relation = \(relation))

//Third, show that the 3-parameter do: works…  
 print ("\nOne triple per line, version1 of relation is")

relation do {a,b,c in print ("\n(\(a.terseDescription) \(b.terseDescription) \(c.terseDescription))")}

//Fourth, show that the 1-parameter do: works…  
 print ("\nOne triple per line, version2 of relation is")

relation do {triple in print ("\n(\(triple.0.terseDescription) \(triple.1.terseDescription) \(triple.2.terseDescription)) ")}

}

static func **example2** () -> Void {

//Relation.example2()

//First, build a relation.

var relation<Int,String> = Relation (from ([(2 "<" 3) (1 "=" 1)

(3 ">" 1) (2 "<" 4)(1 "<" 5) (5 "<" 6) (2 "<" 5) (1 "<" 6) (1 "<" 7)]).

//Second, print the relation…"

print ("\nLet relation = \(relation))

//Third, show that from:relationsDo: works…  
 print ("\nStarting from {1 2 3},'.

Relation.from ([1 2 3]) relationsDo: {relationship, subrelation in  
 print ("\nThe class of the subrelation is \(Utilities.className(of: subrelation))).   
 print ("\nThere is a relationship \(relationship) with subrelation") subrelation do {triple in print ("\n \(triple)}}

//Without debugging information and for different from sets.

for fromCollection in [[1 2 3][1 2][2][]] {

relation from (fromCollection) relationsDo: {relationship, subrelation in

print ("\nThere is a relationship \(relationship) with subrelation") subrelation do {triple in print ("\n \(triple) }}

}

}