```swift
//
//  PointsAndLines.swift
//  Arm
//
//  Created by Erik Nordlund on 4/3/19.
//  Copyright © 2019 Erik Nordlund. All rights reserved.
//
//  Arm Controller includes the following open-source components:
//      • swiftBluetoothSerial: https://github.com/hoiberg/SwiftBluetoothSerial
//      • peertalk-simple: https://github.com/kirankunigiri/peertalk-simple

import Foundation


struct Point2D {
    init(x: Double, y: Double) {
        self.x = x
        self.y = y
    }
    init(x: String, y: String) throws {
        // checking x coordinate
        if let xCoordinate = Double(x) {
            self.x = xCoordinate
        } else {
            if x == "" {
                throw PointError.xStringEmpty
            } else {
                throw PointError.xStringNotANumber
            }
        }

        // checking y coordinate
        if let yCoordinate = Double(y) {
            self.y = yCoordinate
        } else {
            if x == "" {
                throw PointError.yStringEmpty
            } else {
                throw PointError.yStringNotANumber
            }
        }
    }

    func isEqual(to: Point2D) -> Bool {
        if to.x == self.x && to.y == self.y {
            return true
        } else {
            return false
        }
    }
```

```
    func isContinuous(with: Path2D) -> Bool {
        if let nextPoint = with.segments.first?.pointA {
            if self.isEqual(to: nextPoint) {
                return true
            }
        }

        return false
    }

    var x: Double
    var y: Double
}

enum PointError: Error {
    case xStringEmpty
    case yStringEmpty
    case zStringEmpty

    case xStringNotANumber
    case yStringNotANumber
    case zStringNotANumber

    case xOutOfBounds
    case yOutOfBounds

    case xCoordinateOutOfBounds
    case yCoordinateOutOfBounds
    case zCoordinateOutOfBounds

    case nilInput
}

struct Point3D {
    init(x: Double, y: Double, z: Double) {
        self.x = x
        self.y = y
        self.z = z
    }
    /*
     init(fromPoint2D: point2D) {
     self.x = point2D.x
     self.y = point2D.y
     }
     */
    var x: Double
    var y: Double
    var z: Double
}

struct Line2D {
```

```
    init(pointA: Point2D, pointB: Point2D) {
        self.pointA = pointA
        self.pointB = pointB
    }
    var pointA: Point2D
    var pointB: Point2D
}

struct Line3D {
    init(pointA: Point3D, pointB: Point3D) {
        self.pointA = pointA
        self.pointB = pointB
    }
    var pointA: Point3D
    var pointB: Point3D
}

enum LineError: Error {
    case unknownError

    // point A errors
    case xStringEmptyPointA
    case yStringEmptyPointA
    case zStringEmptyPointA

    case xStringNotANumberPointA
    case yStringNotANumberPointA
    case zStringNotANumberPointA

    case xOutOfBoundsPointA
    case yOutOfBoundsPointA
    case zOutOfBoundsPointA

    case nilInputPointA

    // point B errors
    case xStringEmptyPointB
    case yStringEmptyPointB
    case zStringEmptyPointB

    case xStringNotANumberPointB
    case yStringNotANumberPointB
    case zStringNotANumberPointB

    case xOutOfBoundsPointB
    case yOutOfBoundsPointB
    case zOutOfBoundsPointB

    case nilInputPointB
}
```

```
func getLine2D(withBoundsFromZeroTo: Point2D, fromPoint: Point2D, toPoint:
 Point2D) throws -> Line2D {

    do {
        try checkPoint2D(withBoundsFromZeroTo: withBoundsFromZeroTo, point:
         fromPoint)

    } catch {
        //debugPrint("ERROR (Point A): \(error)")
        switch error {
        case PointError.xStringEmpty:
            throw LineError.xStringEmptyPointA
        case PointError.yStringEmpty:
            throw LineError.yStringEmptyPointA
        case PointError.xStringNotANumber:
            throw LineError.xStringNotANumberPointA
        case PointError.yStringNotANumber:
            throw LineError.yStringNotANumberPointA
        case PointError.xOutOfBounds:
            throw LineError.xOutOfBoundsPointA
        case PointError.yOutOfBounds:
            throw LineError.yOutOfBoundsPointA
        case PointError.nilInput:
            throw LineError.nilInputPointA
        default:
            throw error
        }
    }

    do {
        try checkPoint2D(withBoundsFromZeroTo: withBoundsFromZeroTo, point:
         toPoint)



    } catch {
        //debugPrint("ERROR (Point B): \(error)")
        switch error {
        case PointError.xStringEmpty:
            throw LineError.xStringEmptyPointB
        case PointError.yStringEmpty:
            throw LineError.yStringEmptyPointB
        case PointError.xStringNotANumber:
            throw LineError.xStringNotANumberPointB
        case PointError.yStringNotANumber:
            throw LineError.yStringNotANumberPointB
        case PointError.xOutOfBounds:
            throw LineError.xOutOfBoundsPointB
        case PointError.yOutOfBounds:
```

```
            throw LineError.yOutOfBoundsPointB
        case PointError.nilInput:
            throw LineError.nilInputPointB
        default:
            throw error
        }
    }

    return Line2D(pointA: fromPoint, pointB: toPoint)

}

func checkPoint2D(withBoundsFromZeroTo: Point2D, point: Point2D) throws {
    // checking x coordinate
    if 0 ... withBoundsFromZeroTo.x ~= point.x {
        // checking y coordinate
        if 0 ... withBoundsFromZeroTo.y ~= point.y {
            return
        } else {
            throw PointError.yOutOfBounds
        }
    } else {
        throw PointError.xOutOfBounds
    }

}


func getPointFromStrings2D(xString: String?, yString: String?) throws ->
 Point2D {
    if xString != nil && yString != nil {
        do {
            let point = try Point2D(x: xString!, y: yString!)

            return point
        } catch {
            throw error
        }
    } else {
        throw PointError.nilInput
    }
}

func getPointErrorString(forError: Error) -> String {
    return "The coordinates didn't work."
}

func getLineErrorString(forError: Error) -> String {
    return "The coordinates didn't work."
}
```