

## Week 11 Practical Class Exercises

### Lists in Java

---

#### Objectives

The aim of these exercises is to give you some practice using lists in Java. The focus is on the `ArrayList` class and the “for each” loop, but you can experiment with other approaches if you feel confident. You may also wish to discuss Assignment 3 with your tutor.

#### List of Integers

The Java class `ListInts` was discussed in lectures to demonstrate some simple list processing algorithms, similarly to those we studied earlier in Haskell. The `ListInts.java` file is available on the course website.

##### Exercise 1 (Count occurrences)

There is a method `Boolean elem(Integer n)` in `ListInts.java`, which checks whether an integer `n` is in the list. Write a method

```
Integer count(Integer n)
```

which returns a count of the number of times the integer `n` occurs in the list.

You may wish to experiment using *DrJava*, but you should also modify the `TestList.java` class to test your function. Think carefully about your choice of tests. What happens if `n` doesn't appear in the list, or if the list is empty?

##### Exercise 2 (Double each element)

In Haskell, we defined a function to double each element of a list of integers as follows

```
double :: [Int] -> [Int]
double xs = map (*2) xs
```

Add a function to the Java `ListInts` class

```
ListInts times2()
```

which constructs a *new* list of integers, where each element on the result list is twice the corresponding element of this `ListInt` object. (We can't call the method “double” because that's a reserved word in Java.)

Check your implementation, either using *DrJava* or by extending `TestList.java`.

## The Path and Point classes

Java classes `Point` and `Path` respectively representing points in a cartesian coordinate system and paths as a list of points, were discussed in lectures and can be found on the course website. The following exercises are to extend these classes, with the emphasis again being on processing lists and the “for each” loop. As you attempt each of these exercises you should also add some tests to the `TestPoints` class.

### Exercise 3 (Scale and rotate paths)

The `Point` class has a method to translate a point an  $x$ -distance and a  $y$ -distance. `Path` has a corresponding method to translate a path, by translating each point.

Add similar methods to **scale** points and paths by a specified factor. For example, scaling point  $(1.0, 2.0)$  by a factor of  $3.0$  changes it to  $(3.0, 6.0)$ .

Add methods to **rotate** points and paths a specified angle. The formula for rotating point  $(x, y)$  an angle  $\theta$  is  $(x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$ . The trigonometric functions are in `java.lang.Math` so you will need to write `Math.sin(theta)`, for example. Like Haskell, angles are in radians.

### Exercise 4 (Is a path a cycle?)

A path is a cycle if its first and last points are equal. Add a method

```
public Boolean isCycle()
```

to class `Path` to check if a path is a cycle. (Remember the difference between the `(==)` operator and the `equals` method.)

**A different version:** Since our coordinates are of type `Double`, they are only approximate representations of real numbers. Therefore it may be more appropriate to check whether the distance between the points is sufficiently small, say less than  $10e-6$ .

### Exercise 5 (Closest to the origin)

Add a method to the `Path` class which uses the `distance` method in `Point` to determine which point in a path is closest to the origin.