sOFTWARE 2 PRACTICAL

## Classes & Objects

Week 3 – Practical 3

For this week practical, you should create a Java project and write all your code in the **src** folder. You may remember the exercise we have done in SOF1 week 8 and 9 regarding vectors. For your convenience I have rewritten the definition here.

A vector of dimension can be represented by an array of double in Java. We would like to create a class Vector with two basic operations on vectors:

Scalar product:

Addition:

Implementing a Vector Class in Java

### Exercise 1: Class’s constructor

Create a package sof2week03softwarelab, and within this package create a public class Vector. The next step is to define what will be the internal representation of a vector and then write the public constructor Vector(double[]). The design decision is to store the element of the vector in an double array {a,b,c}, for example the vector in an double array {1.0,3.2,7.0}. The constructor will take only one parameter, an array of double. The instance attribute vector. should have a **copy** of the array passed in the parameters.

### Exercise 2:

Another very useful method to write is String toString(). This will enable us to print the content of the instance using the System.out.print methods. For the purpose of this exercise we have decided to represent the vector with the String ”[a, b, c]” .

Implement the static method main to write some tests, that is create instances of the class Vector and print them on the console.

Adding behaviours to the class Vector

We now need to think about the definition of a vector, what operation could be done? We know that we can add two vectors of same dimension, we can do the scalar product with a number (called a scalar), what else?

* Get the dimension of a vector (e.g. the number of elements in the vector),
* Get the value at a defined position in the vector,
* Set a value at a defined position in the vector,
* Check if they are equals, not equals,
* Do the scalar product,
* Do an addition between two vectors of equal size.

### Exercise 3:

Implement the **method** int size() that returns the dimension of a vector (i.e. the number of elements in a vector)

### Exercise 4:

Implement the following accessor and mutator:

* Double get(int index) which returns the value of the element at position index in the vector
* Double set(int index, double value) which set the element at position index to the new value value. The method returns the previous value at position index.

### Exercise 5:

Implement the scalar product method Vector scalarProduct(double scalar). The method needs only one parameter, the scalar. In addition, the method should return a **new** Vector containing the result of the operation, but MUST NOT modify the calling instance, e.g. myVector.scalarProduct(3.0) must not modify the instance myVector.

### Exercise 6:

Implement the method Vector add(Vector other) that emulate the vector addition operator. The method should return a new Vector object.

* You must check that both vectors have the same dimension, return null if it is not the case.
* You must return a new Vector instance like we have done in the method   
  scalarProduct(double)

### Exercise 7:

In Programming, being able to compare objects is important, in particular determining if two objects are equal or not. Let’s try a comparison of two vectors:

double[] data = {1.0,2.0,3.0};

Vector vector1 = new Vector(data);

Vector vector2 = new Vector(data);

vector1 == vector2; // is false

vector1 != vector2; // is true

vector3 = vector1;

vector3 == vector1; // is true

As you can see, in the current state of implementation of our class Vector does not produce the expected result when comparing two vectors. In the example above the == operator return true if the two vectors are physically stored at the same memory address, it does not compare the content of the two vectors.

Therefore, you need to implement a method boolean equals(**Object** other) that returns true if other is a vectors equals to the calling instance (i.e. have the same value at the same position), false otherwise. Note the type of the parameter is **Object**, not **Vector**.

Java provides the following rules to override equals method Java. As per thefollowing rule equals method in Java should be:

1. Reflexive: that is an object must be equal to itself.
2. Symmetric: if a.equals(b) is true then b.equals(a) must be true.
3. Transitive: if a.equals(b) is true and b.equals(c) is true then c.equals(a) must be true.
4. Consistent: multiple invocations of equals() method must return the same value until any of their properties are modified. So, if two objects are equals in Java they will remain equals until any of their property is modified.
5. null comparison: comparing any object to null must be false and should not result in NullPointerException. For example a.equals(null) must be false.

**Hint**: to check if an object is of a certain type, you can use obj instanceof Class. For example, other instanceof Vector.

**Note:** in Python we were able to overload the operators ==, + and \* using the special methods \_\_eq\_\_, \_\_add\_\_ and \_\_rmul\_\_. This is not possible in Java. We can define any name for the methods emulating the + and \* operators, however for the == we don’t have the choice, we must implement the method:

public boolean equals(Object other).