Exercises

IShape exercise

This is very similar to what we did with vehicles and brands…

Implement classes :

- Square
- Rectangle
- Circle
- Shape
- Polygon
- Quadrilateral

Needed Methods :

- **Constructors** {square and circle can be created with a 3 parameters constructor only : (x,y,size) because we need (square.width == square.height)}
- **double getX()** {can be called on : (Square,Rectangle,Circle,Shape,Polygon)}
- **double getY()** {can be called on : (Square,Rectangle,Circle,Shape,Polygon)}
- **double getWidth()** {can be called on : (Square,Rectangle,Circle,Shape,Polygon)}
- **double getHeight()** {can be called on : (Square,Rectangle,Circle,Shape,Polygon)}
- **double getArea()** {can be called on : (Square,Rectangle,Circle,Shape,Polygon), implement accordingly : (circle.getArea() != square.getArea())}
- **int getNumberOfSides()** {can be called on (Square,Rectangle,Circle,Polygon) implement accordingly (Circle.numberOfSides() == 0)}

Think ahead

Try making a quick sketch of how you would implement the above classes taking into account all methods and fields needed. Try avoiding any duplicate of code (Same function with same return statement in two or more different files)

After you are done implementing them…

Go check the solutions package, compare your with my solution and see if you managed to write less code lines than me :p .

I will also leave a diagram of how the classes look like in their hierarchy (within my solution).

INumber exercise

This is also very similar to what we did in FindMin, but this time the INumber interface extends the java.util.Comparable interface, which is similar to the Compare interface we have seen in the assignment. An interface extending another interface it's like combining the methods of both interfaces in the (subclass) interface, therefore, when implementing the interface INumber, it's like implementing INumber,Comparable.

By default, the INumber interface returns the logic for isEqualTo, isLessThan, isGreaterThan, but we must implement the function *int compareTo(E other)* which is inherited from the Comparable interface to return the following:

- 0 if this == other
- -1 if this < other
- 1 if this > other

I also left a NumberTest main class in the ExercisesCompare folder… You may see that it works until we test StringNumber (which I already Implemented) but you need to make it work for DoubleNumber and IntegerNumber make sure you are able to find the right interface implementation before checking the solutions

IList exercise

Here we have to implement a LinkedList which is an important data structure: You can check more in detail what a linked list is here : this article explains well the concept of data structures , pointers and how lists are implemented.

LinkedLists are just one of the infinitely many existing implementations of a list, and it may be more useful in some cases rather than others.

Implement methods

- **E first()** (returns the first element without removing it)
- **E removeFirst()** (removes the first element, then returns it)
- **E getItemAt(int index)** (find the element at the imaginary position 'index' and return it if exists)
- **void add(E element)** (add an element at the default position, therefore as the last element)
- **int size()** (returns the size of the list)

Note:

To properly implement the LinkedList data structure all you need is **a single field** for the FIRST node and a **counter** integer field. NOTHING ELSE !

The Node class is an auxiliary class which contains the element you want to store at that node, and a **pointer** to the next node. The concept of pointer is explained in the article above. I mentioned imaginary position index in the functions descriptions because LinkedLists are not supposed to store the index position of each node, but you can easily define an order in the list by going from first to first.next , to first.next.next and so on… until this is not null