Technische Universität Braunschweig

iRMB

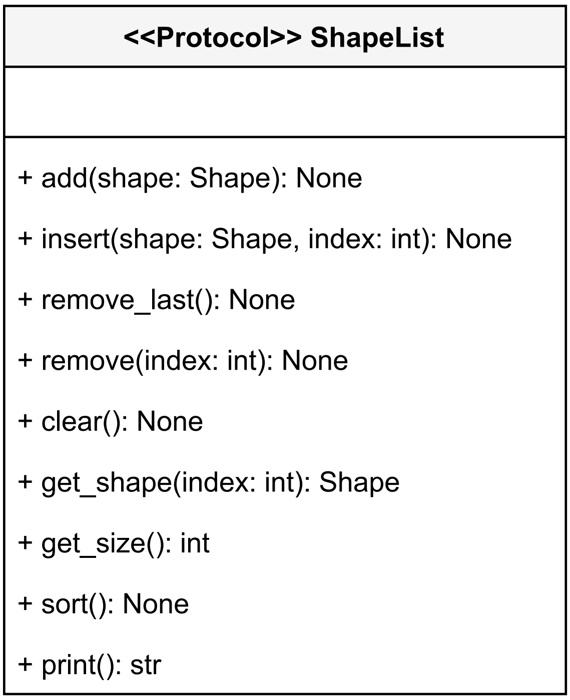
Institut für rechnergestützte Modellierung
im Bauingenieurwesen
Prof. Dr.-Ing. Henning Wessels

Assignment 02 - ShapeLists:

Your task is the implementation of two different lists which both can store objects of the type *Shape*. The class *Shape* has been defined in Exercise 4, which means you also have to complete the tasks from this homework for this assignment.

One of lists you have to implement should be based on a linked list and the other should be based internally on an *numpy array*. However both lists should offer the same functionality which is required by the *protocol ShapeList*.

The following UML-diagram the *protocol*, which both lists are required to inherit.



<<Protocol>> ShapeList

+ add(shape: Shape): None
+ insert(shape: Shape, index: int): None
+ remove_last(): None
+ remove(index: int): None
+ clear(): None
+ get_shape(index: int): Shape
+ get_size(): int
+ sort(): None
+ print(): str

You can find an initial project which already contains the *Shape* classes, the *ShapeList protocol* and the starting point for the *ShapeLinkedList* (including the *Node*) and *ShapeArrayList* classes.

Short description of required functionality:

add(shape: Shape): None Adds the passed *Shape* to the end of the *ShapeList*.

insert(shape: Shape, index: int): None Inserts the passed *Shape* to the position described by the passed index.

remove_last(): None Removes the last Shape from the *ShapeList*.

remove(index: int): None Removes the *Shape* described by the passed index from the *ShapeList*.

clear(): None Clears the whole *ShapeList*.

get_shape(index: int): Shape Returns the *Shape* described by the passed index.

get_size(): int Returns the size of the *ShapeList*.

sort(): None Sorts the *ShapeList* according to the area size of each *Shape* starting with the *Shape* with the smallest area.

print(): str Prints the content of all elements on the console by invoking the *__str__()*-function on all elements.

For the implementation of the sort-method you can choose any sorting algorithm from the lecture. Pay attention to methods whose behaviour changes depending on the state of the list. In addition, it is **mandatory** that user inputs leading to an error are catched (e.g. trying to remove an element at position 10 while the list only contains 5 elements) and to inform the user of the incorrect input. This needs be achieved by *raising an IndexError*:

```
raise IndexError("Your message goes here")  # automatically stops the function
```