Specifications: `PostFilterMappingTransformer`

A `PostFilterMappingTransformer` object first filters a `List<LabeledDouble>` object and then applies a map to each element of the filtered collection to obtain the elements of the collection it returns. Such an object can be used for a variety of purposes. One example is to create a `List<LabeledDouble>` that contains the credits earned for each course that a student passed from a `List<LabeledDouble>` of course grades. For example, given:

```
List<LabeledDouble> grades;
```

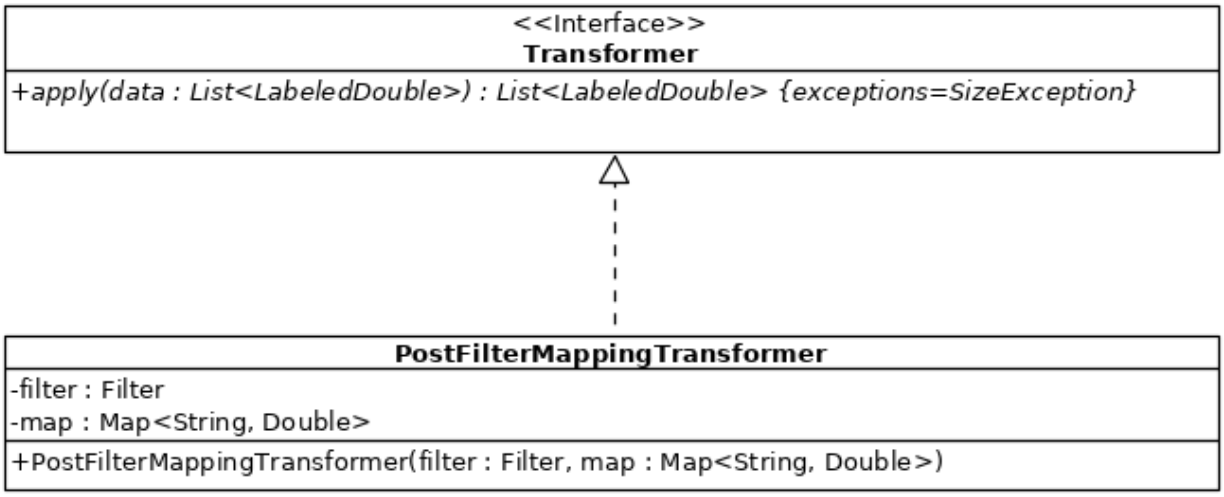
that contains the grades of a student for each course (where the course identifier is the label), and:

```
Map<String, Double> courseMap;
```

that contains a mapping from course identifiers to credits, one could accomplish this as follows:

```
Filter passing = new ThresholdFilter(0.0, 1);
Transformer transformer = new PostFilterMappingTransformer(passing,
                                                           courseMap);
List<LabeledDouble> credits = transformer.apply(grades);
```

The following UML class diagram illustrates the relationship between the `Transformer` interface and the `PostFilterMappingTransformer` class.



In addition to the obvious specifications illustrated in the UML class diagram, the `PostFilterMappingTransformer` class must satisfy the following specifications.

1. You may assume that the `apply()` method is passed a `List` that does not contain any `null` elements.
2. The `apply()` method must not have any side effects. That is, it must not change the parameters that it is passed in any way and it must not change any attributes in any way.
3. The `apply()` method must construct a new `List` that is a subset of the `List` it is passed.
   3.1. If the `apply()` method is passed a `null` `List` then it must throw a SizeException.
   3.2. If the `apply()` method is passed a non-`null` then it must return a new `List` or throw a `SizeException` as described below.
      3.2.1. The `List` to return must be calculated by first applying the `filter` attribute (if it is non-`null`) to the original `List` and then applying the `map` attribute to the filtered `List`. (If the `filter` attribute is `null` then the original `List` must not be filtered in any way.)
         3.2.1.1. Each element in the returned `List` must be a `LeafLabeledDouble`.
            3.2.1.1.1. The `label` attribute must be the `label` of the element in the filtered `List`.
            3.2.1.1.2. The `value` attribute must be the `value` in the `map` object that has the same key as the `label` attribute.
               3.2.1.1.2.1. If the `map` object is `null` then the `value` attribute must be `null`.
               3.2.1.1.2.2. If the `label` attribute does not have a corresponding key in the `map` object then the `value` attribute must be `null`.
      3.2.2. If the `List` to return contains no elements then the `apply()` method must throw a `SizeException`.