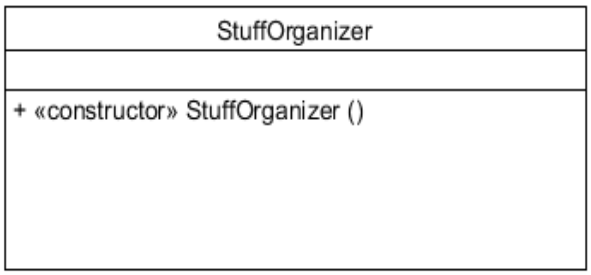
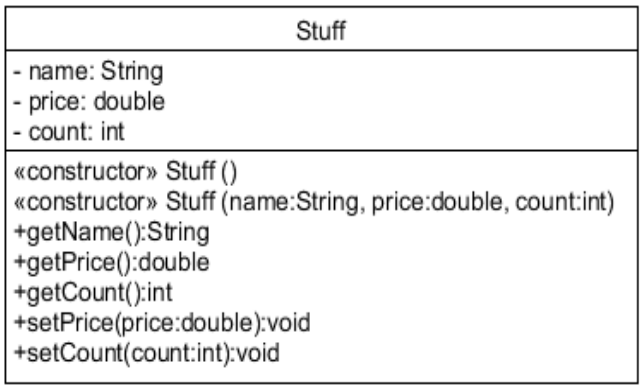
A Friendly Warning:
By copy-pasting, you deprive yourself of the learning process, the exercise is designed to give you. Copy-paste only when the exercise asks you to do so.

Stuff Organizer

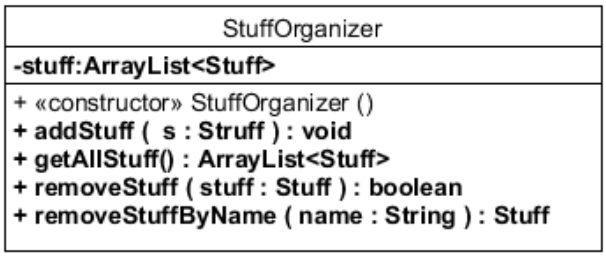
Review Exercise: for-each, while, ArrayList, 1-* + **new**: for-loops & arrays

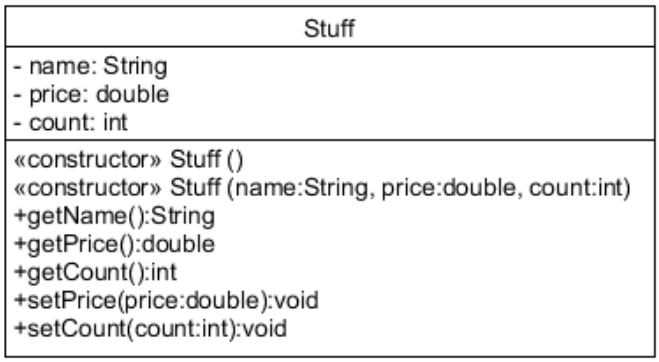
1. Implement the UML class diagram



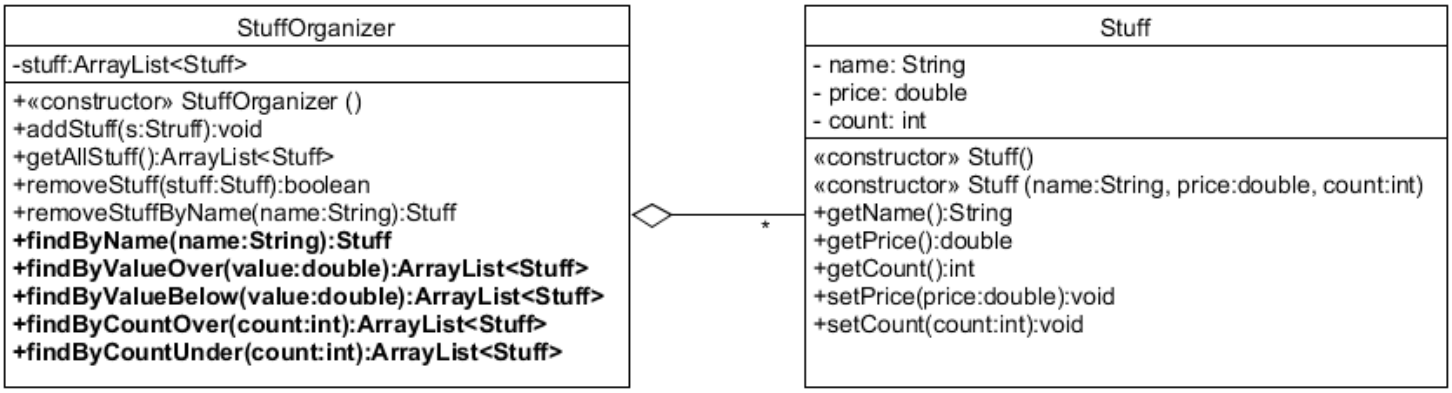


2. Implement the association between the two classes and the related methods (shown in bold). Consider whether a for-each or a while should be used in each case.





3. Implement other relevant functionality (shown in bold face). Take care to **use a for-each loop where possible**, otherwise you should use a **while** loop. **Assume unique Stuff name attributes!**

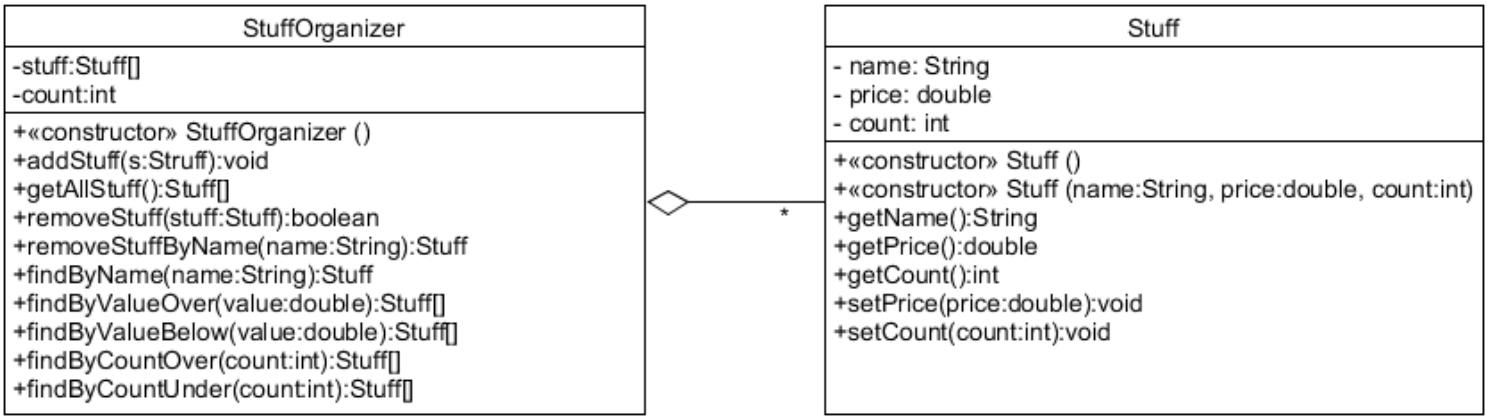


3.a. Make sure that no duplicate Stuff names are allowed in the same StuffOrganizer. (Reject the adding of Stuff that has a name that is already in the StuffOrganizer.)

Solved all of this? You are doing really well in Programming! Stuck halfways through exercise 3? You are still doing well, keep up the good work!

4. For-loops: Replace all loops (for-each loops and while loops with **for-loops**). *You may want to save the BlueJ project under a new name for reference!*

5. Advanced exercise: Replace the ArrayLists with arrays all over the place.
Exercise 5 may be (very) difficult if you are a beginner, but you may find it an interesting challenge if you have substantial prior coding experience.



Pre: Copy the BlueJ project or just duplicate the classes s.t. you have the orignial classes from exercises 1 – 3 (or 4) for fallback.

a. Change the attribute that implements the association, add the **count** attribute.
   count on StuffOrganizer keeps track of how many Stuff objects have been added to the array.
b. Change the constructor in StuffOrganizer
c. Change the add method to implement a range check (don't exceed the capacity)
d. Implement a private method that converts an ArrayList to an array of Stuff.
   Alternatively, look up the toArray(…) method on ArrayList
e. Change all the methods that return ArrayLists to return arrays of Stuff
   (use the result of 5.d to implement this functionality)

Notes to exercise 5:
- When you solve this exercise, make sure that you have a copy of your previous work, as you may want to go back and check your original solution. Also, changing the original project makes you lose your previous work.
- In your working copy, you may find it easier to comment the class body and impelement the old methods one-by-one. This way you can keep your code in a compilable state and you can keep testing your work. Use **`/* code...code...code */`**
- What should happen if you try to add more Stuff than you can contain in your array?
- What should happen when you remove Stuff from the array? Leave a **null** value?
- Maybe a private method could clean up the hole left after removing a Stuff object.
- To implement a conversion method, you can do one of two things – I suggest, you do both just for the fun of it (and to learn from it):

5.d: This is how you get started with 5.d:
1. You create an array of the desired size, loop through your ArrayList and add the elements one-by-one to the array.
2. You use the built-in conversion method:

```
Stuff[] stuff = stuffArrayList.toArray(new Stuff[0]);
ArrayList<Stuff> list = new ArrayList<>(Arrays.asList(stuff));
```