SHARE-APP CODE SMELLS :

1. DUPLICATED CODE :

CODE SMELL LOCATION :

- **loadItems(Context context)** method in **ItemList** class and **loadContacts (Context context)** in **ContactList** class
- s**aveItems (Context context)** method in **ItemList** class and **saveContacts (Context context)** in **ContactList** class

Why I'm considering this as code smell :

In the above locations I had identified **duplicated code** smell. Block of code exists in the loadItems and loadContacts methods in ItemList and ContactList classes are almost similar with very slight differences. Same for saveItems and saveContacts methods in ItemList and ContactList classes. This fits the description of duplicated code smell "*Duplicated code occurs when blocks of code exist in the design that are similar, but have slight differences*"

Why this has to be fixed :

This can be a problem, because if something needs to change may be an implementation pattern (algorithm) for loading and saving contacts or items, then the code needs to be updated in multiple places (So at this point we have to update in ItemList and ContactList code, in future if we are having multiple lists then we have to update new algorithm or pattern in multiple places).

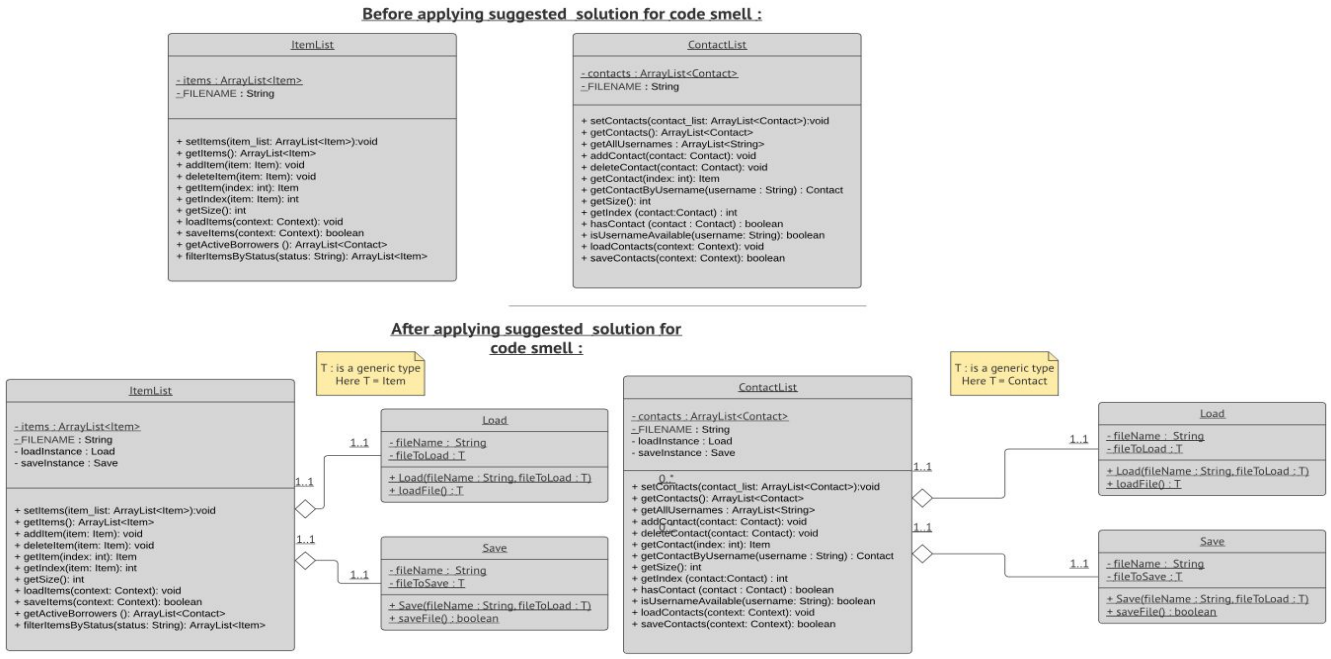
Instead, if the code only needed to be updated in one location, it is easier to implement the change. It also reduces the chance that a block of code was missed in an update or change.

Solution :

We can use aggregation here, So I had separated Load and Save Logics into individual classes that take typeOfClass(generics) and fileName(string) as parameters for the constructor of these new classes.

So now if we want to change Load or save procedure then we have to update only in the new classes and thereby we can also implement **separation of concerns** . Now ItemList and Contact List classes can use Load and Save functionalities using Load and

Save classes.



2. LARGE CLASS

CODE SMELL LOCATION :

- ItemList and ContactList classes.

Why I'm considering this as code smell :

I'm considering this as a "Large class " code smell not only because these classes cross 100+ lines of codes, but it also handles **multiple responsibilities** more than they should do. This fits the "*Large classes occur when more responsibilities are needed, and these classes seem like the appropriate place to put the responsibilities, Over time, however, these responsibilities might attract more responsibilities, and the class continues to get larger and larger and take on more and more responsibilities.*" large class code smell's description.

Why this has to be fixed :

Over time, however, these responsibilities might attract more responsibilities, and the class continues to get larger and larger and take on more and more responsibilities and eventually it may end up with a god class kind of structure.

This growth will require extensive comments to document where in the code of the class certain functionalities exist. Comments, as discussed previously in this lesson, are another indicator of bad code. It is helpful to catch this code smell early, as once a class is too large, it can be difficult and time-consuming to fix.

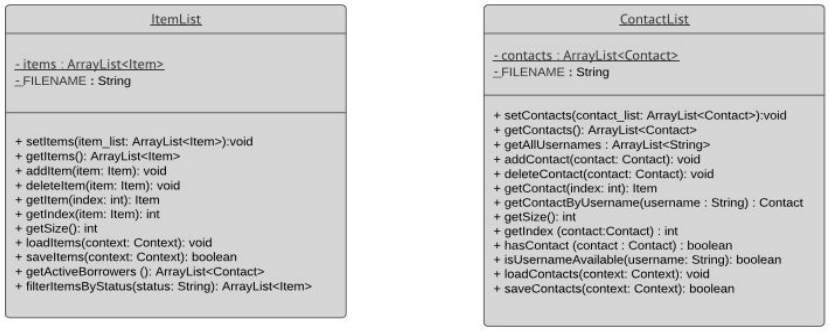
Solution :

“Classes should have an explicit purpose to keep the class cohesive, so it does one thing well. If a functionality is not specific to the class’ responsibility, it may be better to place it elsewhere.”

In this context loadItem & saveItem methods in ItemList and loadContact & saveContact methods in contactList were handling responsibilities that are not relevant to these classes. So we can separate these logics to separate classes to handle loading and saving responsibilities.

In the below diagram I had separated (splitted) load and save responsibilities to Load and Save classes.

Before applying suggested solution for code smell :



After applying suggested solution for code smell :