In the Retriever package (in Controller) there is an abstract class called “Retriever.” It has one public method, “getAll(),” which returns a list of DatabaseObjects. Campuses, Ministries, Events, and everything else we might want to store in the database are all DatabaseObjects (this may change in a few cases but it will remain true for anything we would want to show in a Card). DatabaseObjects have methods to get their name, description, image (not working right now), and their unique database id. It also has a HashMap of fields to values (all Strings), which will allow for more customized values (Events have a “time” and Campuses don’t). Once the view has these objects, it can display them as a card.

To get these objects, each Fragment that wants to display them as cards will extend “CruCardScreen” (name up for interpretation, probably should have ‘fragment’ in there). This class extends CRUFragment and doesn’t change much for now, but has get/set methods for a Retriever. This is where it pulls the objects to display. This is nice because the fragment logic never touches the model; it doesn’t even need to know what type of DatabaseObject it has, and we can switch out retrievers whenever we want. This will help with standardization of the UI interface and simplicity/modularity in the code. From the UI standpoint, this is all it sees of Retrievers.

Retriever also has some protected (invisible outside it’s package so it isn’t confusing), abstract methods that subclasses must implement. These return a “Class” object, and a String. The Class method returns the class of the type of object being instantiated (Ministry.class, Campus.class, etc.), and the String method returns the string from the strings.xml file that needs to be passed to the database to get all of that type of object. All of the logic needed to get and return a list of those objects is in Retriever’s getAll() method, so the children can be very simple, with only two, one-line methods. These children are what we can swap in and out of the UI fragments.

Why have children at all if they’re so simple?

* If we kept all of the logic in the parent class (by having a method to get all Campuses, another for Ministries, etc.) we would have reduced modularity because we couldn’t easily delete everything relating to one type of object. Retriever would also be big and ugly.
* This allows us to switch which objects are displayed in the UI without changing any UI code. Some changes would be required if the UI used the hashmap of fields, but the generic card-displaying code wouldn’t.
* Children *CAN* be complicated (yay! :D). They can filter or sort, and remove the need for any of that code in the UI (unless we resort in the screen, then it should be handled there).
  + For example, if we wanted to show a screen in the initial setup that showed all ministries for a campus, we could write a retriever for that. If we changed our minds later and wanted to show all ministries, regardless of campus, we could write another one (only one line of code difference!) and swap them back and forth whenever we want.

**Short version:**

UI People:

* Make your fragment a CruCardScreen, pass it an appropriate retriever, and call getAll() to get your objects.

Non-UI People:

* Make children of Retreiver, put any filtering logic in there, or anything else you can think of.
* Don’t touch the reflection.

Here’s a picture:

