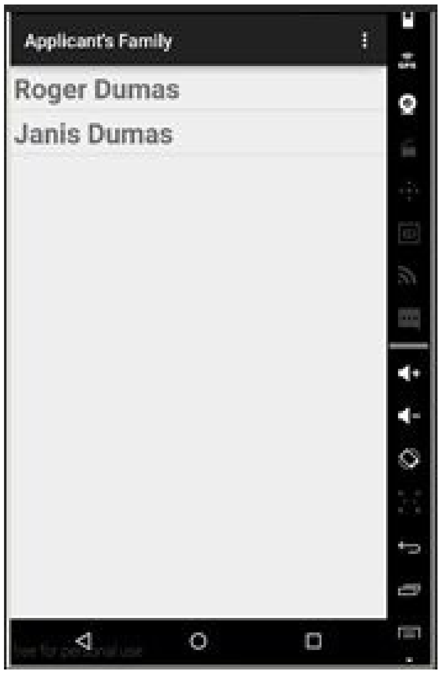
ACTIVITY **2.2.3**

# ListView

## INTRODUCTION

You've likely used apps that include a `ListView` like the one shown below. The default "Contacts" app is probably the most commonly used `ListView` on a phone. A `ListView` displays as many items as can be shown on the screen, and allows for scrolling to view more items, if any are available.



In this activity, you will add a `ListView` to the College App to display multiple family members. For now, the family members all belong to the `Guardian` class.

### Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student

**RESOURCES**

**Activity 2.2.3 Visual Aid**
Resources available online

**Free Response Question: MediaItem**
Resources available online

## Procedure

### Part I: The Family

1. In Android Studio, open your CollegeApp from *Activity 2.2.2 Remote Database*. as directed by your teacher. (If you use the solution code, change `BE_APP_ID`, `BE_ANDROID_API_KEY`, and `MY_EMAIL_ADRESS` in *`ApplicantActivity.java`* to reference your personal Backendless and email values.)

With the goal of displaying the applicant's family members in a list, it makes sense to create a data structure to hold all of the data for these family members.

2. Create a new class called `Family`.
3. Give this class a `TAG` for logging purposes.

**NOTE**

You may use `<ClassName>.class.getName()` to retrieve the name of a class instead of using a `String` literal.

4. Add an `ArrayList` to the `Family` class to contain data of type `FamilyMember`. Identify this `ArrayList` with the name `family`.
5. Go to the API for Java and research `ArrayList`. What methods do you think will be of most use to you?

   `Family` will be a **singleton**.

   The "s" prefix is used for `static` class variables. Recall from *Lesson 1.1*, a static variable means there is exactly one `sFamily` variable created for this class. As a singleton, `sFamily` will reference the `Family` object that gets created, meaning it references itself or it is self-referencing. If this seems redundant, it is. When you create the `Family` class, its reference is saved as the single, static `sFamily` variable, thus guaranteeing that one and only one class is created, known as a singleton.

6. To accomplish this, add a `private` and `static` variable identified by `sFamily` of type `Family`.

The `Family` class will need to have an instance of the `ArrayList` class to store `FamilyMember` data.

7. Create a constructor for the `Family` class without any parameters. A constructor without parameters is called the **default constructor**. In the default constructor, assign a new `ArrayList` to the private instance variable `family`. In a singleton, only the class itself should have access to its constructor, to avoid making multiple copies; so make this constructor `private`.

8. Within `FamilyMember`, add a constructor that takes as its parameters, a first name and a last name, and stores them in the appropriate fields.

9. Within `Guardian`, add a similar constructor with parameters for the first and last name, taking advantage of the `super` keyword.

10. Now you will populate `family` with some test data: In the constructor for `Family`, make two calls to the `ArrayList` `add` method to add elements to the `family` instance variable:

    a. In the one call to `add`, pass a `Guardian` object created using the `Guardian` constructor that takes two `String` parameters (you may use `String` literals here).

    b. In the second call to `add`, pass a new `Guardian` object created using the default constructor. This will indicate an error.

    The error you see applies to `Guardian's` constructor. The error occurs because you changed how `Guardian` is constructed in step 9. Before this, Guardian did not have *any* constructor. Whenever it needed to instantiate, the `Guardian` class was constructed or instantiated with the constructor of its *closest parent*, in this case, `FamilyMember`. If a class has no constructor, the JVM (Java Virtual Machine) instantiates it using the constructor of its closest parent. For classes without an explicit parent, the JVM will use `Object's` constructor as the default constructor.

    Now that Guardian *has* a constructor, the default constructor is no longer automatically provided, and you must provide a default constructor.

11. Create a default constructor in `Guardian` that will invoke the constructor of its parent, `FamilyMember`.

```
public Guardian() {
    super();
}
```

You will now access the two `Guardians` you just added to your `Family`.

12. To access the data contained in a singleton, `Family` needs a `get` method. Create a `public`, `static` method with return type `Family`, identified by `get`, that takes no parameters.

13. To the new `get` method, add a conditional to prevent a `NullPointerException` from occurring if `sFamily` is `null`.
14. Within the `get` method, return `sFamily`.
15. If `sFamily` is `null`, assign it a `new` instance of the `Family` class.
16. Add the getters and setters for the `ArrayList family`.

## Part II: Displaying the Family Using ListView

Until this part of the activity, you couldn't run your code and see it work within your app.

17. Create a new class called `FamilyListFragment`.

    This class will replace `GuardianFragment` in the `NavigationDrawer`. However, don't delete `GuardianFragment` yet; you will need it later to show the detailed view of a `Guardian` when the user taps on the guardian in the `ListView FamilyListFragment`.

18. Make `FamilyListFragment` extend `ListFragment`. Import `android.support.v4.app.ListFragment`.
19. Add a `TAG` to the class for logging purposes.
20. Add a member variable of type `Family` called `mFamily`.
21. Create a default constructor for `FamilyListFragment`.
22. To create and store the singleton reference to the `Family` class, within the constructor you created in the last step, call `Family's static get` method and assign the resulting instance to `mFamily`.

```
mFamily = Family.get();
```

23. Add the `onCreate` method and appropriate string resource to display "Applicant's Family" as shown below.

```
@Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      getActivity().setTitle(R.string.family_members_title);

      FamilyMemberAdapter adapter = new FamilyMemberAdapter
      (mFamily.getFamily());
      setListAdapter(adapter);
  }
```

Until we create a `FamilyMemberAdapter` class in the next step, lines 6 and 7 will generate errors.

24. Add the following inner class to `FamilyListFragment`. For the `View` import, import the Android View and *not* the Backendless View.

```
private class FamilyMemberAdapter extends ArrayAdapter
<FamilyMember> {
   public FamilyMemberAdapter(ArrayList<FamilyMember> family) {
       super(getActivity(), 0, family);
   }

   @Override
   public View getView(int position, View convertView,
                       ViewGroup parent) {
       if (convertView == null) {
           convertView = getActivity().getLayoutInflater()
                  .inflate(R.layout.list_item_family_member,
                          null);
       }

       FamilyMember f = getItem(position);

       TextView nameTextView =
                 (TextView)convertView
                          .findViewById(R.id.family_member_
                                       list_item_nameTextView);
       nameTextView.setText(f.getFirstName() + " " +
                            f.getLastName());

       return convertView;
   }
}
```

There will be additional errors until you create the appropriate resource file in the next step.

25. Create a layout file named ***list_item_family_member*** that has a `TextView` as its root element. Give this `TextView` an `id` attribute with value `family_member_list_item_nameTextView`. Add the following attribute to this `TextView`: `android:textSize="30sp"`.

26. Create a ***string.xml*** resource for `family_members_title` with a value of "Applicant\'s Family".

27. Test your app. What do you notice?

**Check your answer**

No changes yet! Needs to be wired up to the Navigation Drawer.

28 Modify your navigation drawer as appropriate so the Family List displays in place of the Guardian screen. Reference *Activity 2.1.4* if necessary.

29 Run and test your app.

## Part III: Static Variables (AP Focus)

You have seen a variety of static methods and static constants throughout these activities. In your next Free Response Question (below), you will explore a new type of variable, a **static variable**. Similar to static constants, static variables are created within a class. Each static variable has *exactly one* storage location for all instances (objects) created from the class and all objects share it and the data stored there. However, static variables have one important difference from static constants: the value of a static variable can change.

**static variable**

Also called a **class variable**, a static variable is a variable that is shared across all instances (objects) of the class. All instances have access to the value stored in the variable.

Why is this useful? Static variables are useful when all instances of a class need access to some information that needs to change. For example, a static variable can be used when you need a unique ID number. Think of your student ID, or perhaps your social security number. This is a unique number for the "instance" of you. When you start school you are usually assigned a student ID, so we'll use this as an example. Consider a `Student` class:

```
public class Student {
    private String firstName;
    private String lastName;
    private static int ID = 1001;

    public Student(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
        ID++;
    }

    @Override
    public String toString() {
       return ID + ":" + firstName + " " + lastName;
    }

    // include a main for easy testing
    //(this breaks abstraction for the class!)
    public static void main(String[] args) {
       System.out.println(new Student("Frodo", "Baggins"));
       System.out.println(new Student("Samwise", "Gamgee"));
       System.out.println(new Student("Gandalf", "the Gray"));
   }
}
```

When you construct a `Student`, `ID` is incremented. Since `ID` is a `static variable`, all `Student` objects share the same storage location for `ID` ; when one `Student` object increments the `ID`, the other instances see this increment. Compare this to the instance variables `firstName` and `lastName`. Each `Student` object has their own copy of `firstName` and `lastName` and do not share it with the other instances of `Student`.

30 Copy and paste the `Student` code into a new BlueJ project and class called `Student`. Since this class has a `main` method (intentionally breaking abstraction) you can-right click on `Student` to run it directly and omit a Tester or Runner class. After you compile and run it once, change the value of `ID` in your code and re-run your program. Notice how *all* student objects share this `ID` and  update it.

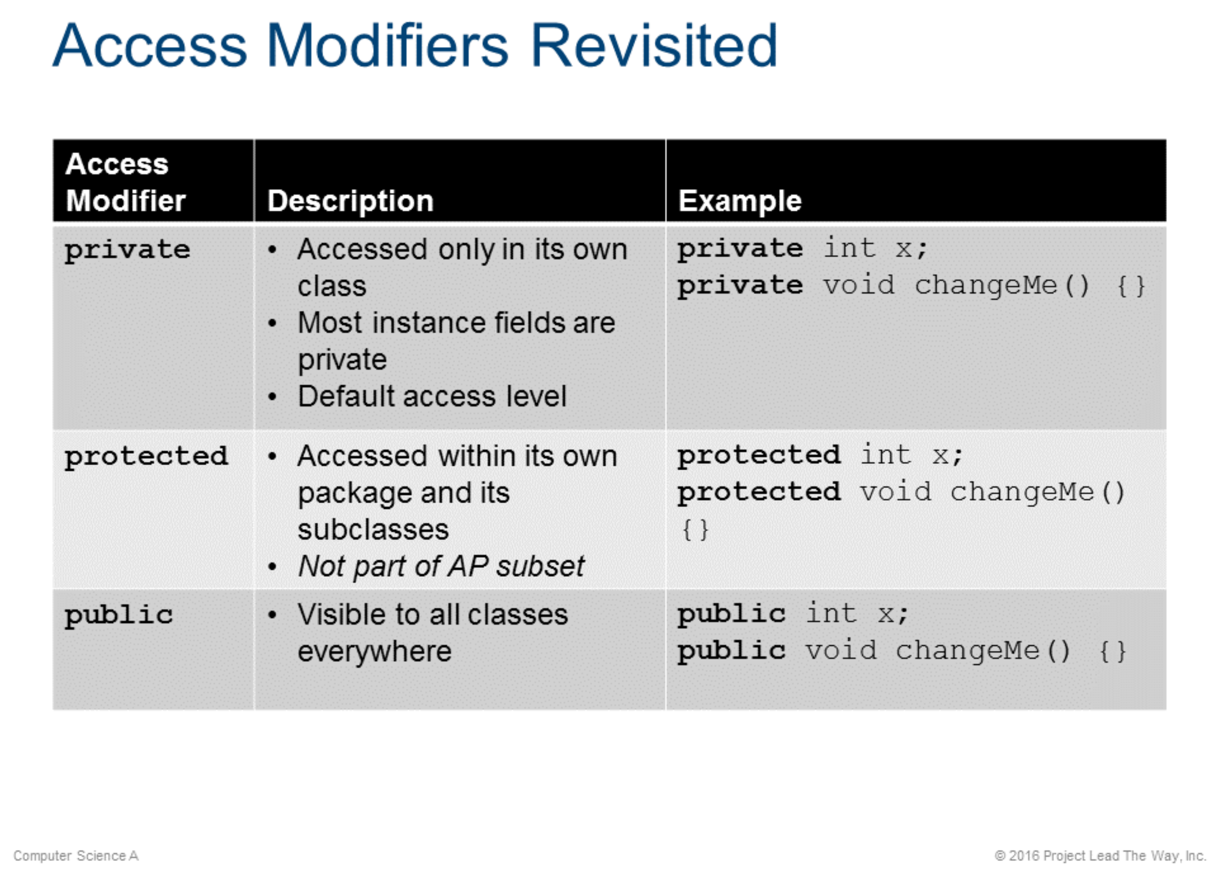
**AP Focus:** **FRQ: MediaItem**

31 Once you have completed the MediaItem FRQ, create a *MediaItem* project and class in BlueJ. Replace the default code with your solution code. Override the `toString` method to show all instance variables and the ID of each media item created. Then, add this at the end of your class to test. If you have answer the FRQ correctly, the ID should increment with each media item created.

```
// include main in your class just to test
public static void main(String[] args) {

     // NOTE: you may need to change the name of the static
     constants GAME, DVD, and CD to match yours
     MediaItem m1 = new MediaItem(MediaItem.GAME, "Portal2",
                    4/19/2011", 49.95);
     System.out.println(m1);

     MediaItem m2 = new MediaItem(MediaItem.DVD, "Young
                    Frankenstein", "12/4/1974");
     System.out.println(m2);

     MediaItem m3 = new MediaItem(MediaItem.CD, "The White
                    Album", "11/2/1968");
     System.out.println(m3);
}
```

## CONCLUSION

1. Summarize what each of the access modifiers do, and how they are used.



### Access Modifiers Revisited

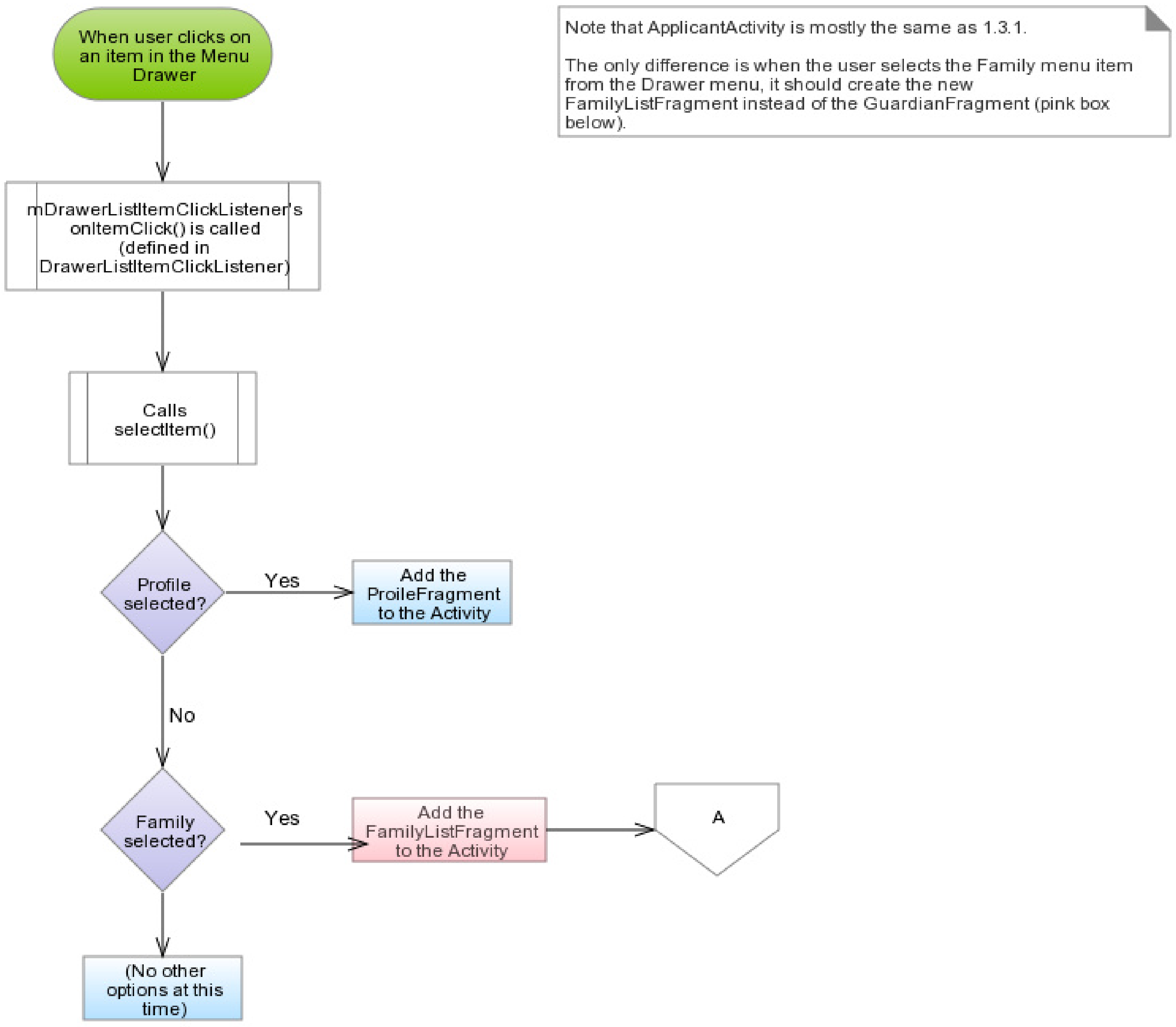
| Access Modifier | Description | Example |
|---|---|---|
| `private` | • Accessed only in its own class<br>• Most instance fields are private<br>• Default access level | `private int x;`<br>`private void changeMe() {}` |
| `protected` | • Accessed within its own package and its subclasses<br>• *Not part of AP subset* | `protected int x;`<br>`protected void changeMe() {}` |
| `public` | • Visible to all classes everywhere | `public int x;`<br>`public void changeMe() {}` |

Computer Science A

© 2016 Project Lead The Way, Inc.

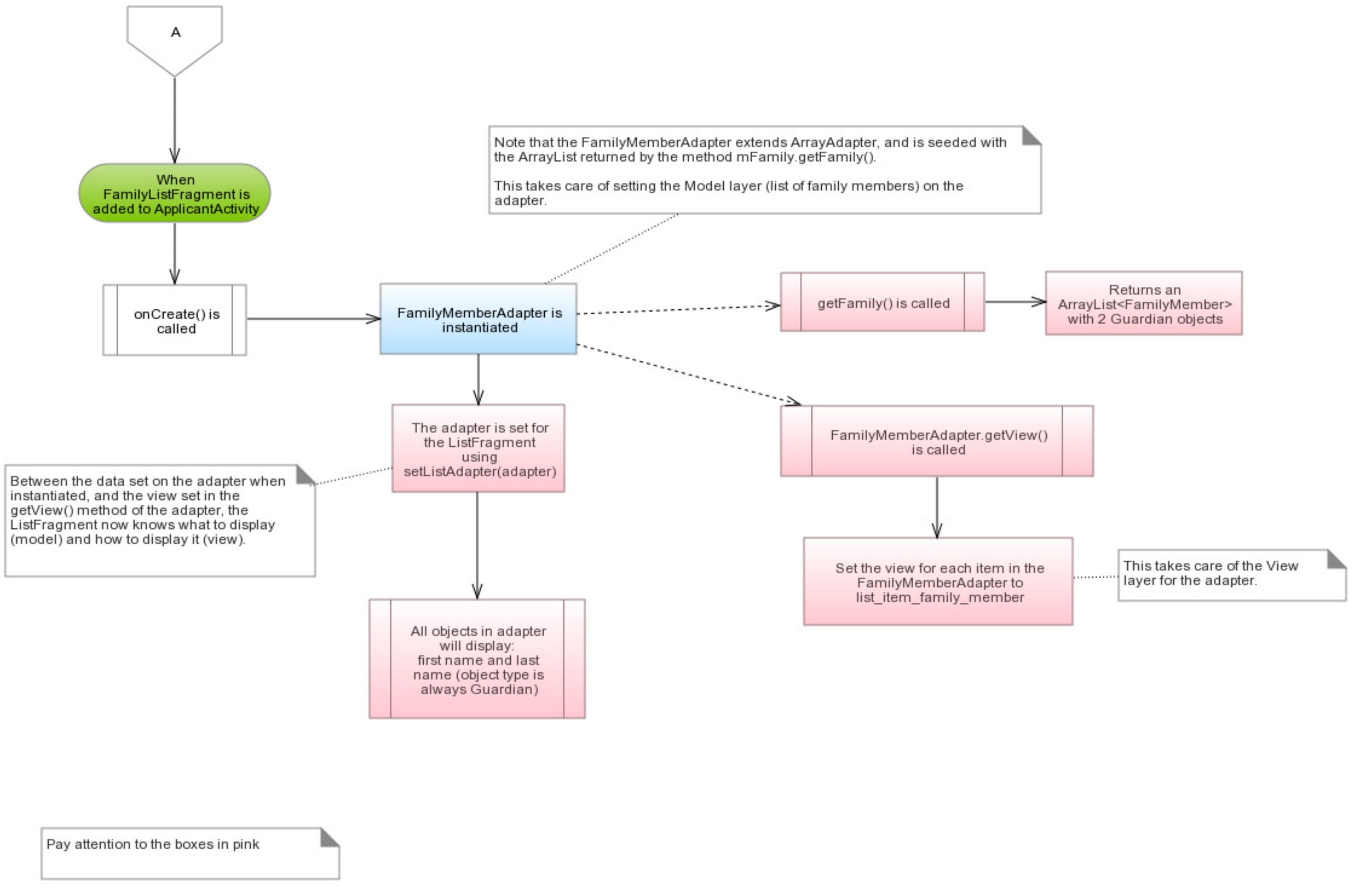
2. Why would you use an `ArrayList` instead of an array for storing the family?

# Activity 2.2.3 Visual Aid


When user clicks on an item in the Menu Drawer
mDrawerListItemClickListener's onItemClick() is called (defined in DrawerListItemClickListener)
Calls selectItem()
Profile selected?
Yes
Add the ProileFragment to the Activity
No
Family selected?
Yes
Add the FamilyListFragment to the Activity
A
(No other options at this time)

Note that ApplicantActivity is mostly the same as 1.3.1.

The only difference is when the user selects the Family menu item from the Drawer menu, it should create the new FamilyListFragment instead of the GuardianFragment (pink box below).


A
When FamilyListFragment is added to ApplicantActivity
onCreate() is called
FamilyMemberAdapter is instantiated
Note that the FamilyMemberAdapter extends ArrayAdapter, and is seeded with the ArrayList returned by the method mFamily.getFamily().
This takes care of setting the Model layer (list of family members) on the adapter.
getFamily() is called
Returns an ArrayList<FamilyMember> with 2 Guardian objects
The adapter is set for the ListFragment using setListAdapter(adapter)
Between the data set on the adapter when instantiated, and the view set in the getView() method of the adapter, the ListFragment now knows what to display (model) and how to display it (view).
FamilyMemberAdapter.getView() is called
Set the view for each item in the FamilyMemberAdapter to list_item_family_member
This takes care of the View layer for the adapter.
All objects in adapter will display: first name and last name (object type is always Guardian)
Pay attention to the boxes in pink