ACTIVITY **3.3.4**

Polymorphic Behavior

INTRODUCTION

In the previous activities, you created a polymorphic list of contacts that consisted of family members, friends, classmates, and workmates. Until now, the polymorphic nature of the list has not been significant, because the only feature of the polymorphic list is a display value on your screen. With some additional features in your app, you will see more polymorphic behavior.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student
- A Google account and a project on the Google API Console

RESOURCES

Activity 3.3.4 Visual Aid
Resources available online

Procedure

Your goal for this activity is to create a variety of dialogs so the user can enter information particular to each type of trip companion. You will create an "information" `ImageButton`, similar to the Delete `ImageButton`, for each companion in the "Companions for this Trip" list. When the user clicks the **Information** button, its listener's `onClick` method will determine the type of companion and display the appropriate dialogs.

Part I: Set Up the New Functionality

1. Open your TripTracker app in Android Studio.

NOTE

If you were unable to complete the *Activity 3.3.3 Persistent People*, import *3.3.3TripTracker_Solution* as directed by your teacher. Recall that if you import the solution, you must update keys values in *strings.xml*:

- Change be_app_id and be_android_api_key to your Backendless App ID and Android API key values, respectively. You can retrieve these from your **Backendless Console** (**Manage** icon).
- Change the value for google_app_id to your Google Play service API key. You can retrieve it from the **Google API Console** (**Credentials**) .

2. In *companion_item.xml*, add the following XML, which adds an Information ImageButton to each item in the companions list. Place this after the TextView:

```
<ImageButton
    android:src="@android:drawable/ic_dialog_info"
    android:id="@+id/companion_info"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleX=".5" android:scaleY=".5"
    android:layout_alignTop="@id/companion_textView"
    android:layout_alignBottom="@id/companion_textView"
    android:layout_alignParentRight="true" />
```

3. Open *TripFragment.java*.

4. In the CompanionsAdapter getView(…) method, add the following code for the new Information ImageButton. You also need to create the mInfoCompanionButton variable of type ImageButton:

```
mInfoCompanionButton = (ImageButton)convertView.
findViewById(R.id.companion_info);
MyInfoCompanionListener infoCompanionListener = new
MyInfoCompanionListener();
mInfoCompanionButton.setOnClickListener(infoCompanionListener);
mInfoCompanionButton.setTag(tripContact)
```

Notice the last line of the code above that invokes a new setTag method. With setTag, you can attach data to a view, in this case, you are attaching the tripContact data to each `mInfoCompanionButton`. Tag data such as this allows you, as the app developer, to specify any kind of data to store with the View object. When the app runs and a user clicks the button, you will have all of the data associated with `tripContact` data available to you.

5 Finally, define the listener class for the `mInfoCompanionButton`:

```
/*
Get the contact associated with the information button
*/
private class MyInfoCompanionListener implements View.
OnClickListener {
    @Override
    public void onClick(View v) {
        Log.d(TAG, ((Contact)v.getTag()).getName());
    }
}
```

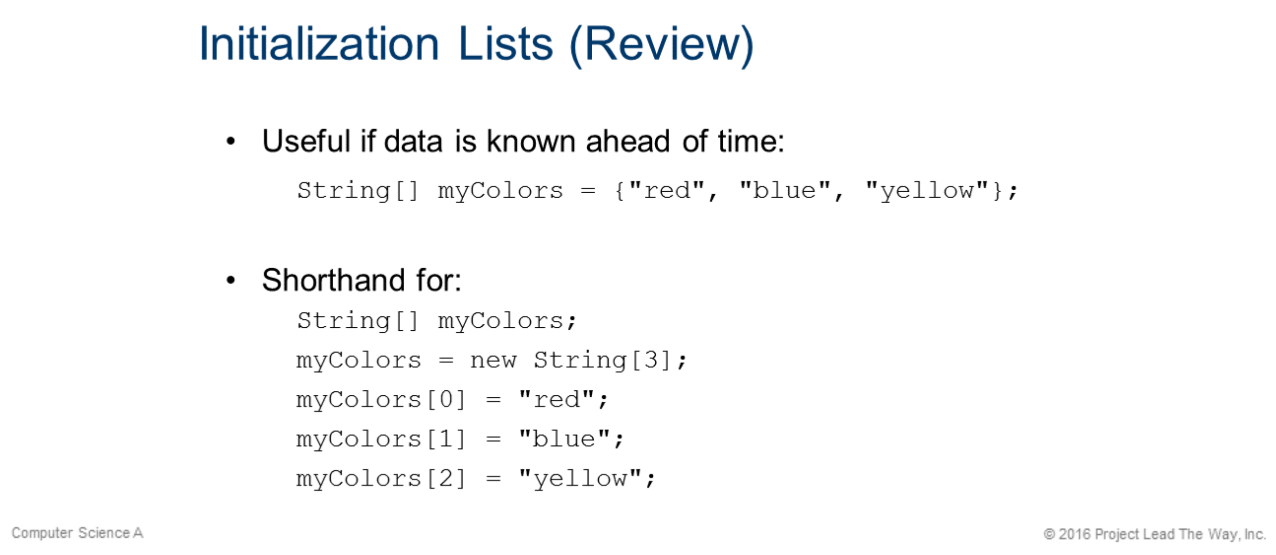
6 Test your app to ensure the listener is working: Click one of the Information buttons and observe logcat to confirm it displays the correct companion.

How does tag data with the Information button?

Check your answer

In the `CompanionsAdapter`, use tag data on the Information button to save the current contact in the list. In this, the button's listener, the parameter v is the Information button so you can retrieve the contact object with `getTag()` and then call the method `getName()` to display the contact's name.

Next you will enter some details about your travel companions. You will need three arrays that contain generic information about family members, classmates, and clubs. For your convenience, here is a quick review about how initialization lists are used in Java:



Initialization Lists (Review)

- Useful if data is known ahead of time:

```
String[] myColors = {"red", "blue", "yellow"};
```

- Shorthand for:

```
String[] myColors;
myColors = new String[3];
myColors[0] = "red";
myColors[1] = "blue";
myColors[2] = "yellow";
```

Computer Science A

© 2016 Project Lead The Way, Inc.

7 Continuing in `TripFragment`, create array class constants `RELATIONSHIPS`, `CLASSES`, and `CLUBS` using **initialization lists**. Some family `RELATIONSHIPS` are “father”, “mother”, “sister”, “brother”. Yours may differ. Similarly, use initialization lists to populate the `CLASSES` and `CLUBS` lists. Take care with the syntax and access specifiers; they should be local constants.

initialization list
A way to declare and initialize an array in one step.

NOTE

Normally, you would not enter the string values directly into your Java source code; rather you would use the *strings.xml*. In this case, the string values more clearly demonstrate how to use an initialization list. Later on, you may choose to define the array in XML.

8 Add the following string definitions to the *string.xml* file:

```
<!—3.3.4 info dialogs -->
<string name="dialog_family_title">Relationship</string>
<string name="dialog_friend_title">How long have you known
this friend?</string>
<string name="dialog_friend_hint">Years</string>
<string name="dialog_work_title">Your workmate</string>
<string name="dialog_work_title_hint">Work Title</string>
<string name="dialog_work_company_hint">Company Name</
string>
<string name="save">Save</string>
<string name="cancel">Cancel</string>
<string name="dialog_classes_title">Classes we took
together</string>
<string name="dialog_club_title">Clubs we share</string>
```

9 Create two XML layout files:

- *dialog_friend_info.xml*

```
<LinearLayout xmlns:android="http://schemas.android.com/
apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/dialog_friend_title"  />
    <EditText
        android:id="@+id/friend_known_for"
        android:inputType="numberDecimal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/dialog_friend_hint" />

</LinearLayout>
```

- *dialog_work_info.xml*

```
<LinearLayout xmlns:android="http://schemas.android.com/
apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

   <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/dialog_work_title" />
   <EditText
    android:id="@+id/work_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginBottom="4dp"
    android:hint="@string/dialog_work_title_hint" />
   <EditText
    android:id="@+id/work_company"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="4dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginBottom="16dp"
    android:hint="@string/dialog_work_company_hint"/>

</LinearLayout>
```

NOTE

These layouts will be used for the dialogs for `Friend` and `Work` contacts. The other types of contacts will use predefined dialogs that do not need layouts defined for them.

In the coming steps, you are going to need the contact type variables (`mContactTypeFamily`, `mContactTypeFriend`, etc.) to determine the type of contacts you have in `TripFragment`. These values are defined in your `ContactFragment` class, inflated from *`strings.xml`* to describe "Family", "Friend", etc. Since this information is now also needed in `TripFragment`, this is a rare circumstance where you make an instance variable `public` for other classes to use. So that other *fragments* can access these variables, they must also be `static`.

10 The best place for these contact type variables is in your `Contact` class. Make the changes in `ContactFragment` and `Contact` so that `Contact` has the four `public static` variables.

11 There is no new functionality yet, but you may want to test your app to be sure all is well.

Part II: Add the Polymorphic Functionality

In `TripFragment` you will modify your new `MyInfoCompanionListener onClick(View v)` method to create a variety of dialogs. The dialogs will get information about your companions, such as "brother" for a family member or "US History" for a schoolmate.

12 Review *3.3.4 Visual Aid* to learn the new execution flow in `TripFragment`.

13 Retrieve the contact that is stored in the view's tag data and save it in a local variable `Contact c`.

If Backendless supported fully polymorphed objects in its database, this contact would have been retrieved from the database as a subclass, such as a `Friend` or a `Family`. Instead, Backendless requires all contacts to be stored as the parent class, `Contact`. But recall that when you updated the trip, you saved its type based on its class name using `newContact.setType()`.

14 Determine the type of contact the listener responded to:

a. Retrieve the contact's type with `c.getType()`.

b. Compare it to the publicly available variables in `ContactFragment`, specifically, `mContactTypeFamily`, `mContactTypeFriend`, `mContactTypeSchool`, and `mContactTypeWork`.

c. Create the `mSelectedCompanion` according to its type. Be sure to provide the contact's name when you construct the proper subclass.

Your listener should now be ready to respond to each type of contact in your list of companions. Please note that you will not be saving the contact as its subclass in

Backendless, only as the `Contact` object. Enhancing the app to save and retrieve contacts as a `Family`, `Friend`, `School`, or `Work` object would be a good extension project.

15 Before you do any further coding to create the dialogs, do the following:

a. Read through steps 16-19 to get an overview of the requirements.

b. Scan the table in step 19; much of the code you will need is provided in this table.

c. Write some high-level pseudocode for what your app will need to do.

16 Based on the type of contact you determined in step 14, make the appropriate changes:

a. If a companion is a family member, display a dialog to let the user specify one of the `RELATIONSHIPS` and then a dialog to choose the contact's birthday. For birthdays, use a `Calendar` object to get the current date; reuse the `Calendar` code that gets the current date in *`DatePickerFragment.java`*.

b. If a companion is a friend, display a dialog so the user may enter a number for "years known" and a dialog to choose the friend's birthday. The code provided allows for double values, such as "2.5" years.

c. If a companion is a schoolmate, display a dialog to let the user select from the list of `CLASSES` and a dialog to select from a list of `CLUBS`. You will need new methods to:

- Add and remove a single club in the list of clubs
- Add and remove a single class in the list of classes

d. If a companion is a workmate, display one dialog to let the user enter the workmate's title and company name.

NOTE

To dismiss these dialogs when you run the app, click or touch the screen behind them.

17 Collect data from the dialog and save it for the appropriate contact. For example, collect data from the birthday dialog and save it for a `Friend` with `setBirthday()`.

18 To confirm you are processing all data properly, add a `showDetails` method to each subclass that displays the information you collect from each contact type. In the listeners, use `Log.d(TAG, mSelectedCompanion.showDetails());` to display the data.

Part III: The Dialogs

19 In the following dialogs table, use the *completed* code for each dialog, based on the type of contact. Then, use the *partially completed* code for each listener method, completing the "to do" algorithm(s) to gather the appropriate data for each contact type. Your teacher will tell you how many dialogs to complete.

Dialog or Listner	Code

family information dialog	`AlertDialog.Builder builder =`
	`    new AlertDialog.Builder(getActivity());`
	`builder.setTitle(R.string.dialog_family_title);`
	`builder.setItems(RELATIONSHIPS, new MyRelationsDialogListener());`
	`AlertDialog dialog = builder.create();`
	`dialog.show();`

family listeners	`private class MyRelationsDialogListener`
	`    implements DialogInterface.OnClickListener {`
	`     public void onClick(DialogInterface dialog, int which) {`
	
	`     // The 'which' argument contains the index position`
	`     // of the selected item`
	
	`     // todo: set the relationship for mSelectedCompanion`
	`        // to the selected item in the relations array`
	`   }`
	`}`

birthday dialog	`Calendar calendar = Calendar.getInstance();`
	`int year = calendar.get(Calendar.YEAR);`
	`int month = calendar.get(Calendar.MONTH);`
	`int day = calendar.get(Calendar.DAY_OF_MONTH);`
	`DatePickerDialog dateDialog =`
	`    new DatePickerDialog(getActivity(),`
	`    new MyBirthdayDialogListener(), day, month, year);`
	`dateDialog.show();`

Dialog or Listner	Code

birthday listener	`private class MyBirthdayDialogListener`
	`    implements DatePickerDialog.OnDateSetListener {`
	`      public void onDateSet(DatePicker view,`
	`                              int arg1, int arg2, int arg3) {`
	
	`            // todo: determine the values of`
	`            // parameters arg1, arg2 and arg3`
	
	`      if (view.isShown()) {`
	`            // todo: save the birthday for mSelectedCompanion`
	`      }`
	`    }`
	`}`

friend information dialog	`AlertDialog.Builder builder =`
	`                  new AlertDialog.Builder(getActivity());`
	`LayoutInflater inflater = getActivity().getLayoutInflater();`
	`builder.setView(inflater.inflate(R.layout.dialog_friend_info,`
	`                  null));`
	`// action buttons:`
	`builder.setPositiveButton(R.string.save,`
	`                  new MyFriendDialogSaveListener());`
	`builder.setNegativeButton(R.string.cancel,`
	`                  new MyFriendDialogCancelListener());`
	`AlertDialog dialog = builder.create();`
	`dialog.show();`

Dialog or Listner	Code

friend listeners for save and cancel	`private class MyFriendDialogSaveListener implements`
	`               DialogInterface.OnClick Listener {`
	`    @Override`
	`    public void onClick(DialogInterface dialog, int id) {`
	
	`        // todo: findViewById for the EditText on the dialog`
	`        // and save the number of years`
	`        // the user entered, such as 4.5 years.`
	`    }`
	`}`
	`public class MyFriendDialogCancelListener implements`
	`               DialogInterface.OnClickListener {`
	`   @Override`
	`   public void onClick(DialogInterface dialog,`
	`                        int id) {`
	`       //do nothing`
	`   }`
	`}`

list of classes dialog	`AlertDialog.Builder builder1 =`
	`         new AlertDialog.Builder(getActivity());`
	`builder1.setTitle(R.string.dialog_classes_title);`
	`builder1.setMultiChoiceItems(CLASSES,`
	`                             null,`
	`                             new MyClassList DialogListener());`
	`AlertDialog dialog1 = builder1.create();`
	`dialog1.show();`

Dialog or Listner	Code

list of classes listener	`private class MyClassListDialogListener implements`
	`        DialogInterface.OnMultiChoiceClickListener {`
	`    public void onClick(DialogInterface dialog,`
	`                        int which, boolean isChecked) {`
	
	`        // each time a checkbox is clicked (either`
	`        // isChecked or !isChecked), this executes`
	
	`        // The 'which' argument contains the index position`
	`        // of the selected item`
	
	`        // todo: if checked, add this class to the list of`
	`        // of classes for mSelectedCompanion`
	`        // based on the value of the checkbox`
	`        // if not checked, remove it`
	
	`        // to dismiss the dialog, touch or click the`
	`        // Trip Details screen behind the app`
	`    }`
	`}`

list of clubs dialog … and listener	*Duplicate the functionality for the "list of classes dialog" and listener, making changes for the title and the items in the list.*

Dialog or Listner	Code

work information dialog	`AlertDialog.Builder builder =`
	`        new AlertDialog.Builder(getActivity());`
	`LayoutInflater inflater = getActivity().getLayoutInflater();`
	`builder.setView(inflater.inflate(R.layout.dialog_work_info,`
	`                null));`
	`// action buttons:`
	`builder.setPositiveButton(R.string.save,`
	`                new MyWorkDialogSaveListener());`
	`builder.setNegativeButton(R.string.cancel,`
	`                new MyWorkDialogCancelListener());`
	`AlertDialog dialog = builder.create();`
	`dialog.show();`

work listeners	`private class MyWorkDialogSaveListener`
	`        implements DialogInterface.OnClickListener {`
	`    @Override`
	`    public void onClick(DialogInterface dialog, int id) {`
	
	`        // todo: findViewById for the EditText on the dialog`
	`        // and save the work title and company name`
	
	`    }`
	`}`
	`private class MyWorkDialogCancelListener`
	`        implements DialogInterface.OnClickListener {`
	`    @Override`
	`    public void onClick(DialogInterface dialog, int id) {`
	`         // do nothing`
	`    }`
	`}`

20 Test your app and fix some common problems/bugs that may arise:

a. Do not enter a value for “years known”.

b. For a schoolmate, when you click an item in the dialog, you add that class or club to the corresponding list in the `School` object. Can you *remove* a class or club when you *uncheck* an item?

21 Continue testing using logcat to confirm all contact types are asking for the correct information and the methods are storing the proper information.

Part III: Regression Test

22 If you choose to implement some of the Bonus Challenges in the next section, return to this regression testing when you have completed your challenges. Otherwise, your TripTracker modifications are complete! Do some regression testing to ensure all is well with your app.

c. Test parts of the app that might have been affected by the new functionality.

d. Test features of the app you may not have used in a while.

e. Try to act like a user who is unfamiliar with the app and make some “mistakes”.

CHALLENGES

Choose one or more of these bonus challenges as directed by your teacher.

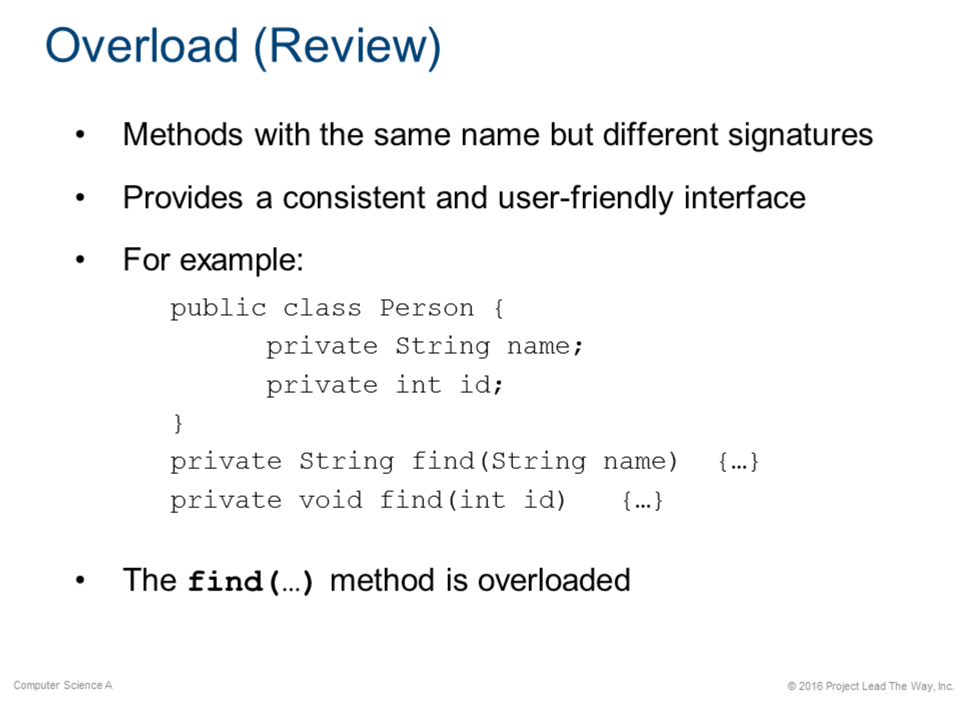
- Some contact names may be too long to fit into space provided in the companions list. Through trial and error, determine the maximum characters that can display. Truncate names that are too long and add an ellipsis (...) to indicate more information exists that cannot be displayed.
- To improve the dialogs for clubs and classes, add a save/cancel option. Use the tutorial at **http://developer.android.com/guide/topics/ui/dialogs.html** to help you code.
- When you navigate back/home from `TripFragment` without saving, your changes are lost. Warn the user that this is about to happen and allow her to confirm or cancel the navigation.

- Another feature that is not implemented is remembering the list of companions you have already selected when you navigate to the list of contacts in `ContactFragment`. Program your app to select the proper radio buttons (Family, Friend, etc.) for all companions when you navigate to `ContactFragment`.
- (*Extra Bonus Challenge*) Create type-specific persistent data for companions in Backendless. Populate the data when the dialogs appear so that the birthday appears on the `DatePicker`, the boss's company name appears, etc.

CONCLUSION

1. The `setBirthday` method in Family and Friend is a good example of polymorphic behavior. Explain how your code demonstrates its polymorphism using this method.
2. Contacts on a device store more information than you used in this lesson, including unique IDs. Two overloaded methods are given for finding a contact.

```
public Contact findContact(String name) {
    for(Contact c : contactList) {
        if(c.getName().equals(name)) {
           return c;
        }
    }
    Log.i(TAG, "No contact found with name " + name);
    return null;
}
public Contact findContact(int id) {
    for(Contact c : contactList) {
       if(c.getId() == id) {
          return c;
       }
    }
    Log.i(TAG, "No contact found with id" + id);
    return null;
}
```



Overload (Review)

- Methods with the same name but different signatures
- Provides a consistent and user-friendly interface
- For example:

```
public class Person {
      private String name;
      private int id;
}
private String find(String name)   {…}
private void find(int id)    {…}
```

- The **`find(…)`** method is overloaded

Computer Science A

© 2016 Project Lead The Way, Inc.

Overloaded functions differ from Overridden methods and the difference can be subtle

Compare

```
// override:
public class Contact {
  private String toString() { }
}
public class Family {
  private String toString() { }   // same signature, overrides and replaces the parent toString method
}
```

with

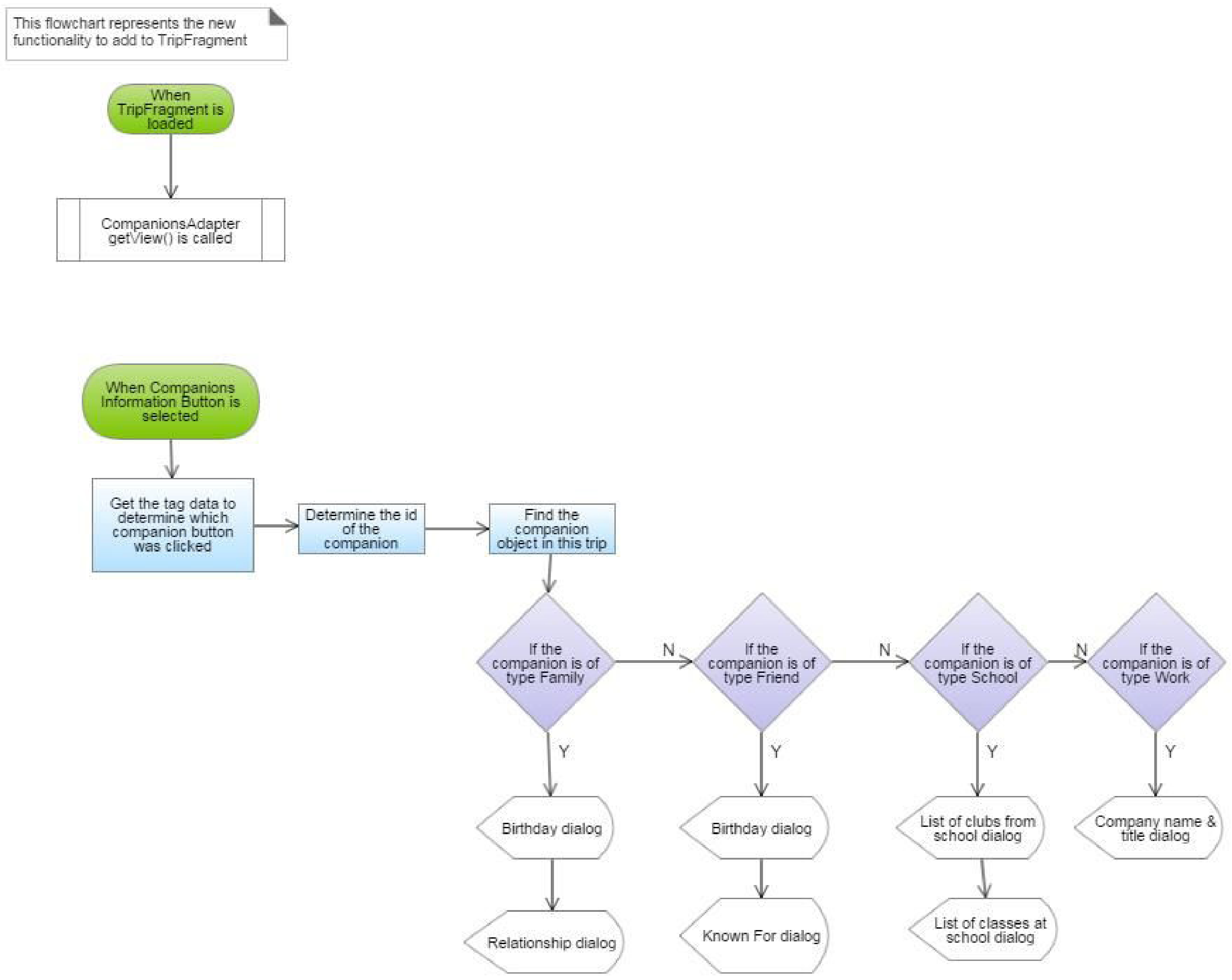
```
// overload:
public class Contact {
  private String toString() { }
}
public class Family {
  private String toString(String whatever) { }   // instead of overriding, the method has a
  // different signature and therefore it overloads
  // results in two different versions of the method
}
```

Review the slide above and explain how the overloading works. Why write an overloaded method instead of a method with a unique name?

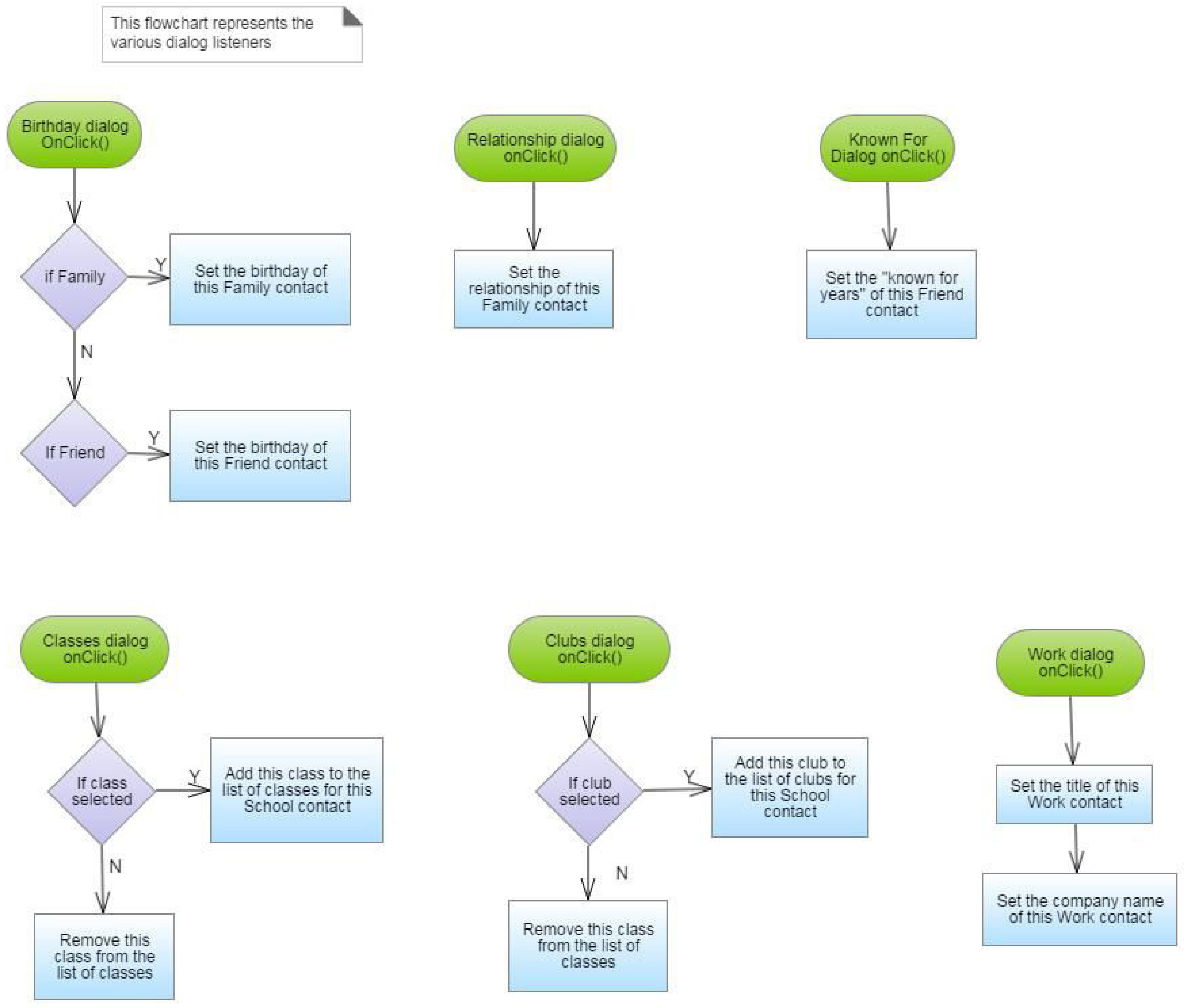
3. What other overloaded functions might be useful for the list of polymorphic contacts?

Activity 3.3.4 Visual Aid

TripFragment


This flowchart represents the new functionality to add to TripFragment
When TripFragment is loaded
CompanionsAdapter getView() is called
When Companions Information Button is selected
Get the tag data to determine which companion button was clicked
Determine the id of the companion
Find the companion object in this trip
If the companion is of type Family
N
If the companion is of type Friend
N
If the companion is of type School
N
If the companion is of type Work
Y
Birthday dialog
Relationship dialog
Y
Birthday dialog
Known For dialog
Y
List of clubs from school dialog
List of classes at school dialog
Y
Company name & title dialog

TripFragment Dialogs


This flowchart represents the various dialog listeners
Birthday dialog OnClick()
if Family
Y
Set the birthday of this Family contact
N
If Friend
Y
Set the birthday of this Friend contact
Relationship dialog onClick()
Set the relationship of this Family contact
Known For Dialog onClick()
Set the "known for years" of this Friend contact
Classes dialog onClick()
If class selected
Y
Add this class to the list of classes for this School contact
N
Remove this class from the list of classes
Clubs dialog onClick()
If club selected
Y
Add this club to the list of clubs for this School contact
N
Remove this class from the list of classes
Work dialog onClick()
Set the title of this Work contact
Set the company name of this Work contact