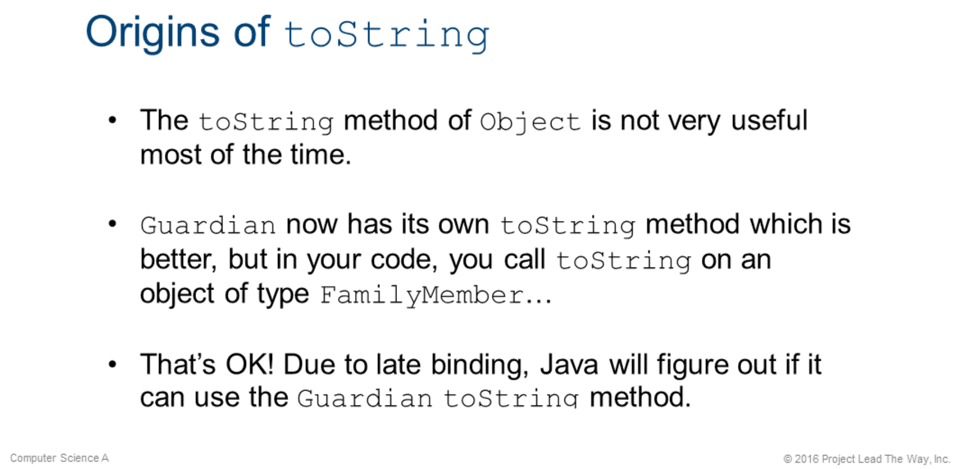
ACTIVITY **2.2.4**

One Method, Many Classes

INTRODUCTION

Polymorphism is a part of object oriented program where an object in your program can reference different data types. Much like a person can be a friend, a brother or a sister, or a son or a daughter, so too can objects represents different data types. At runtime, a method of a polymorphed object might produce different results. With polymorphism, object inheritance and interfaces become really powerful, because they allow you to treat many classes the same way in your code.

polymorphism
The ability of an object to take on multiple data types, all related by the super/sub class relationship.

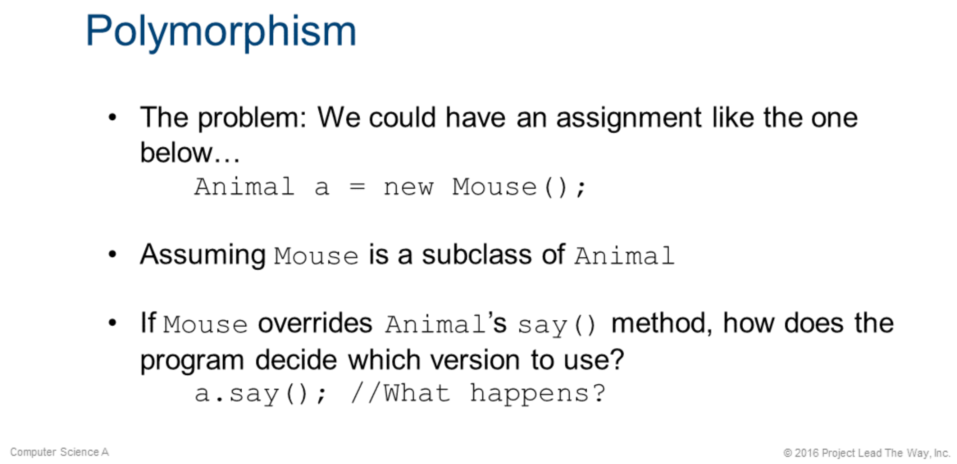


Origins of `toString`

- The `toString` method of `Object` is not very useful most of the time.
- `Guardian` now has its own `toString` method which is better, but in your code, you call `toString` on an object of type `FamilyMember`…
- That's OK! Due to late binding, Java will figure out if it can use the `Guardian toString` method.

Computer Science A

© 2016 Project Lead The Way, Inc.



Polymorphism

- The problem: We could have an assignment like the one below…

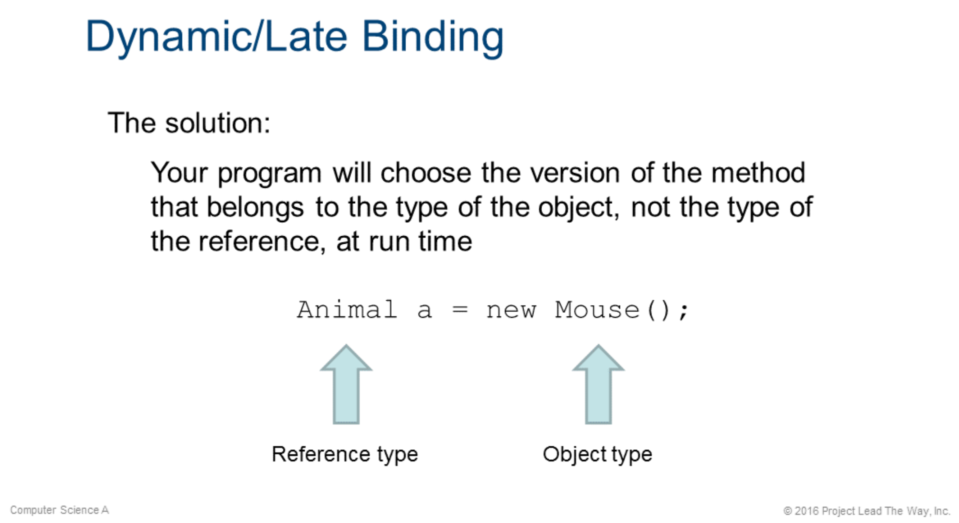
```
    Animal a = new Mouse();
```

- Assuming `Mouse` is a subclass of `Animal`
- If `Mouse` overrides `Animal`'s `say()` method, how does the program decide which version to use?

```
    a.say(); //What happens?
```

Computer Science A

© 2016 Project Lead The Way, Inc.



Dynamic/Late Binding

The solution:

Your program will choose the version of the method that belongs to the type of the object, not the type of the reference, at run time

```
Animal a = new Mouse();
```

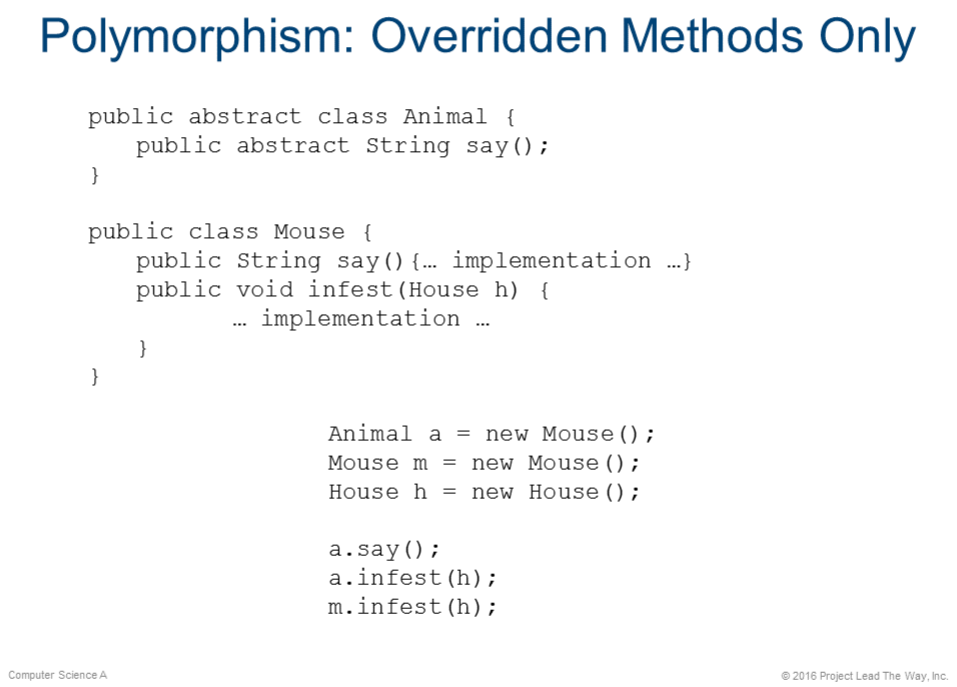
Reference type

Object type

Computer Science A

© 2016 Project Lead The Way, Inc.

At run time, your program will check the type of a, much as we might use the instanceof keyword. It will find that its actual type is Mouse, even though it is referenced by type Animal, and then call the Mouse version of say().



Polymorphism: Overridden Methods Only

```
public abstract class Animal {
    public abstract String say();
}

public class Mouse {
    public String say(){… implementation …}
    public void infest(House h) {
            … implementation …
    }
}
```

```
Animal a = new Mouse();
Mouse m = new Mouse();
House h = new House();

a.say();
a.infest(h);
m.infest(h);
```

Computer Science A

© 2016 Project Lead The Way, Inc.

The catch is that polymorphism only works for overridden methods. Given the two class definitions on the slide, a.say() works fine, a.infest(h) causes a compile time error because Animal a has no awareness of any infest method, but the Mouse's m.infest(h) works fine.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or device emulator

RESOURCES

Activity 2.2.4 Visual Aid
Resources available online

Free Response Question: StringFinder
Resources available online

Procedure

Part I: The Animals Speak

Work through this lesson on **Polymorphism** as directed by your instructor.

1. In Android Studio, open your CollegeApp from *Activity 2.2.3 List View*. If you were unable to finish the activity, open *`2.2.3CollegeApp_Solution`* 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.)
2. Get a copy of *`2.2.4AnimalApp_StarterCode`* source files from your teacher and import the project as directed by your teacher. You should have both projects open in Android Studio at the same time.
3. Run the AnimalApp. What do you see?
4. This app will demonstrate, through a series of text outputs, the power of polymorphism via inheritance. Create an abstract class called `Animal`.
5. Give `Animal` one `public abstract` method `say` with return type `String` and no parameters.
6. Create two classes, `Duck` and `Fish`, each of which are subclasses of `Animal`.

   What keyword do you need to use to make a subclass?

   What method will `Duck` and `Fish` both need to implement?

7. Implement this method by returning the `String` literal `"quack"` for `Duck` and `"blub"` for `Fish`.
8. Add constructors for both `Duck` and `Fish`. The bodies of these constructors can remain empty.
9. All of the code you add to `AnimalActivity` will be after the following line, yet within `onCreate`:

```
output = (TextView)findViewById(R.id.output);
```

10. Instantiate `Duck` and Fish in `AnimalActivity` with identifiers `duck` for `Duck` and `fish` for `Fish`.
11. Add the following two lines after your instantiation.

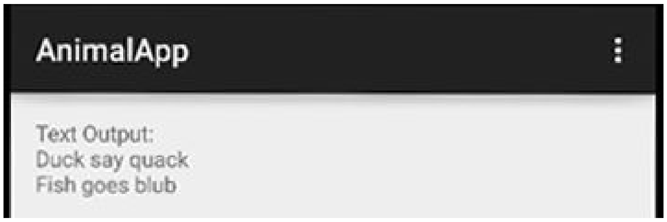
```
this.output.append("\nDuck say ");
this.output.append("\nFish goes ");
```

The escape sequence is useful to help keep your output clear. What does it do?

Check your answer

Escape sequence \n makes a new line, or a “line return”. This causes the next output to go on a new line.

12. After the String literal in each of the calls to append (which you added in the previous step), concatenate an appropriate method call.
13. Test your app to verify that you get the following results.



```
Text Output:
Duck say quack
Fish goes blub
```

14. Add a `Seal` class to your app and generate appropriate output.

Now, to see what polymorphism by inheritance can do for you, you will need an `ArrayList`. You will perform a series of operations on this `ArrayList` to explore polymorphic behavior.

15. Create and initialize an `ArrayList<Animal>` identified by `animals`.
16. Add each of your `animals` to animals using the `add` method of `ArrayList`.

17 Making sure that all of your output happens at the end of the `onCreate` method, replace `fish.say()` in your code with `animals.get(1).say()`.

Is your output still the same? Why or why not? If it is not, fix it so that it is.

18 Replace `animals.add(fish);` with `animals.add(duck);`.

How is your output affected? Do you know why?

Continue with the next steps to help you understand what is happening.

19 Add the following statement to find out what's going on:

```
this.output.append("\nThe object at index 1: " + animals.
get(1));
```

What type of object is stored at index 1?

Check your answer

duck

20 Change `animals.add(duck);` back to `animals.add(fish);`.

21 Replace your three output statements that call say methods with one `for` loop that calls `say` on every object in animals. Verify that the correct numbers of output statements are being generated.

It will require extra effort to make the message different (other than the result of the say call) for each animal. Explain what you would need to do to modify your program so that your new output is exactly the same as your old output.

If there is time, make the changes in your code. *Hint: The String method* `lastIndexOf` *might come in handy. Check the Java API for how to use it.*

22 Add a `for each` loop that accomplishes the same output as in the previous step. (If you made a `for each` loop in the previous step, add a regular `for` loop.) Verify that the outputs of the two loops are the same. Which do you like better and why?

23 Create a new interface called `Fun`.

24 Add one method to this interface called `play` that returns a `String`.

You do not need to include an access modifier when creating a method signature in an interface. Why?

Check your answer

An interface is always accessible to the entire class and its subclasses.

25 Modify `Seal` so that it implements the `Fun` interface. Use the String literal `"The seal bounces a ball off its nose."` for this.

26 Add the following statement to `AnimalActivity` within your `for each` loop:

```
this.output.append("\n" + a.play());
```

Is this a problem? If so, remove it and explain why.

27 Add the following code within one of your for loops to get your Seal to play.

```
if (a instanceof Seal) {
    this.output.append("\n" + ((Seal)a).play());
}
```

What do you think the code above is doing?

28 Create a new class `Fox` that extends `Animal` and create an empty, no-argument constructor to it.

29 Modify `Fox` so that it implements `Fun`, using the `String` literal `"The fox chases mice and digs holes!"`.

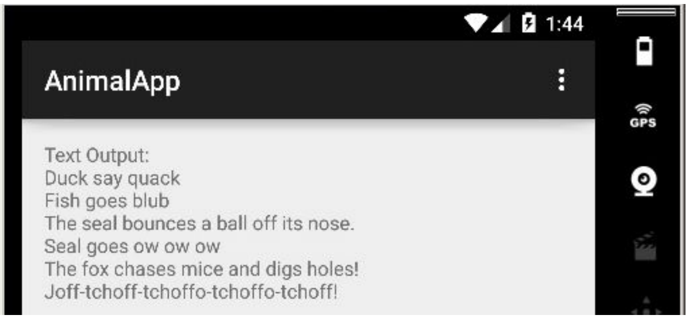
30 Does your `Fox` class have any errors? It should! Why should it have an error?

Check your answer

The error is “Fox is not abstract and does not override abstract method say() in Animal”. Android Studio is reporting that Fox needs to have a defined say method.

31 Fix the error using anything you think a `Fox` would say and add a `Fox` object to `animals`.

32 Modify your `for` loop to produce the output below using as few characters in your String literals as possible.



```
Text Output:
Duck say quack
Fish goes blub
The seal bounces a ball off its nose.
Seal goes ow ow ow
The fox chases mice and digs holes!
Joff-tchoff-tchoffo-tchoffo-tchoff!
```

Why would it be easier to modify the `for each` loop to do this?

AP Focus: **FRQ: StringFinder**

Part II: CollegeApp Uses Polymorphism to Display FamilyMembers

CollegeApp will use polymorphism in its display of `FamilyMembers`. `Guardians` and a new object type, `Sibling`, will both be stored in an `ArrayList` containing data of type `FamilyMember`. Then the app will display the correct information about the object without knowing ahead of time whether it is dealing with a `Guardian` or a `Sibling`.

33. In Android Studio, open your CollegeApp from *Activity 2.2.3 List View*. If you were unable to finish the activity, import *`2.2.3CollegeApp_Solution`* as directed by your teacher.

34. Create a `Sibling` class that is a subclass of `FamilyMember`.

35. Add a `Guardian` constructor that takes a third `String` parameter for `occupation` and set the `occupation` instance variable to the parameter from the constructor. Modify `Guardian(String, String)` and `Guardian()` so that they assign a default value of `"unknown"` to `mOccupation`.

36. Add a `toString` method to `Guardian` that returns an object of type `String`. Format the `String` as follows:

    Guardian: [*first name*] [*last name*]
    Occupation: [*occupation*]

    Now you will check whether your new `toString` method is working. Recall from the last activity, you created an inner class in `FamilyListFragment` called `FamilyMemberAdapter`

37. Modify the `getView` method of `FamilyMemberAdapter` so that instead of displaying the first name of the `FamilyMember` followed by the last name, it calls the `toString` method of the `FamilyMember`.

38. Test your app. The Family Member fragment should show "unknown" occupations for both entries.

39. Define a similar `toString` method for `Sibling` without the occupation.

40. Create constructors that you think will be useful for the Sibling class based on those you find in the `Guardian` class.

41. In `Family`, use one of those constructors to add a `Sibling`. Run and test your app. Does it do what you expect? Add or modify other siblings and guardians.

42. Add the following log statement to the `getView` method of `FamilyMemberAdapter` after the assignment to `f` shown.

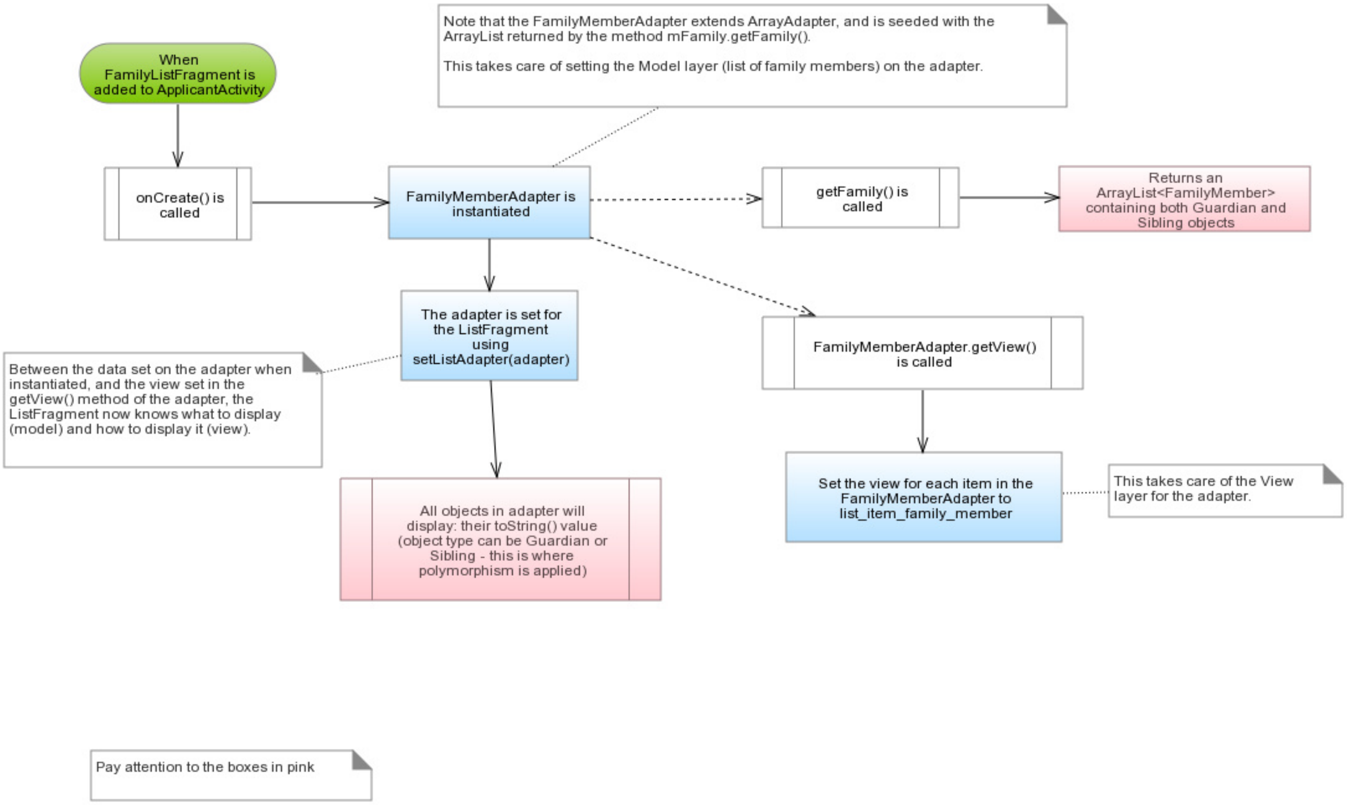
```
FamilyMember f = getItem(position);
Log.d(TAG, "The type of FamilyMember at position " +
position + " is " + f.getClass().getName());
```

What type of object is `f` in each of the log messages when you run the program, and what does this mean?

CONCLUSION

1. Explain in your own words what you think polymorphic behavior is.
2. What is the biggest advantage for creating a polymorphic list, and what is the biggest disadvantage?
3. Compare and contrast: abstract classes vs. interfaces.

Activity 2.2.4 Visual Aid



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> containing both Guardian and Sibling 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).

All objects in adapter will display: their toString() value (object type can be Guardian or Sibling - this is where polymorphism is applied)

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.

Pay attention to the boxes in pink