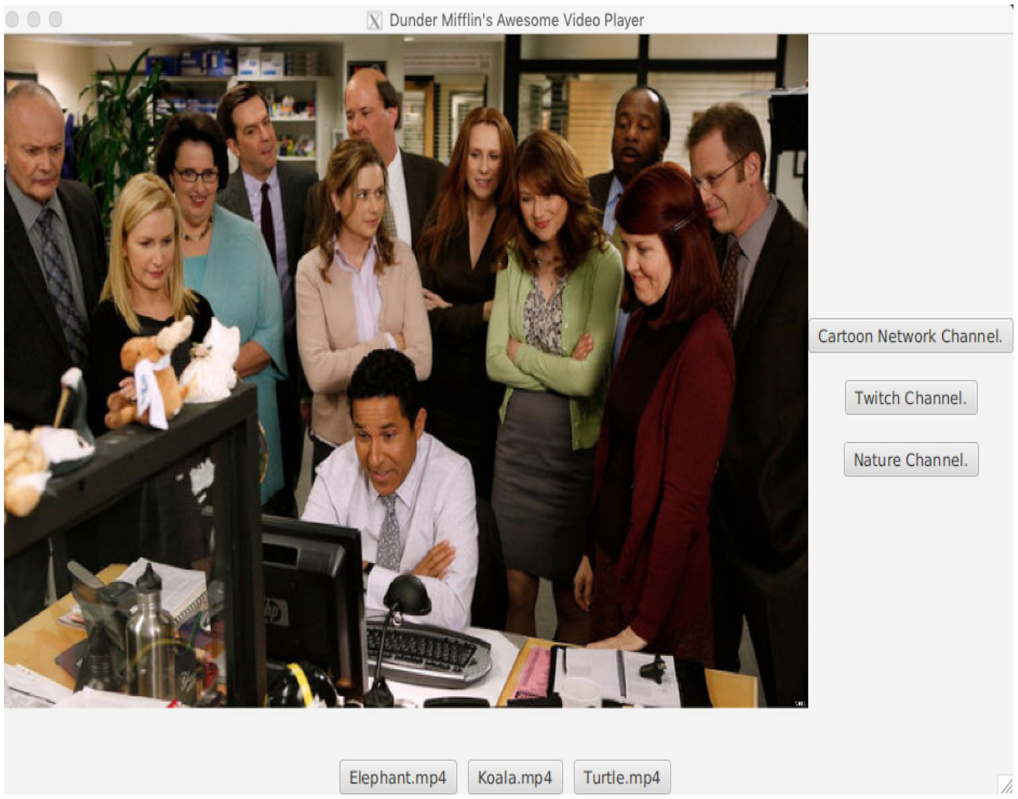
Lab 2: Interfaces and Polymorphism

We've had nearly four weeks of CS15 and by now, the buzzwords of Object-Oriented Programming (OOP) should be familiar to you. OOP is a powerful modeling paradigm that makes programming large scale applications much easier than before. For the remainder of the course, a good understanding of OOP concepts will be integral to designing and implementing the assignments. This lab is about learning how to make your life as a programmer much easier by writing flexible and reusable code through interfaces and polymorphism.

Goal: Learn how to implement polymorphism using an interface. When you're done with this lab, you should end up with a program that looks like this:



Introduction

During the fire drill, Michael sent the office projector out the window, screaming "Help!! Help!!" Unfortunately, projectors were never meant to be dropped from the second story of a building, and now the office needs a new one. Michael sent Toby out to find another projector, knowing that projector salesmen are even more difficult to work with than paper salesmen. After hours on the phone to no avail, Toby becomes frustrated and decides to code his own video player to be used on a computer screen. In this lab, we will use the concepts of interfaces and polymorphism to build a brand new, fully functional video player for the whole office to use.

Because of the support code for this lab, it is very important to follow the directions in the stencil and in the checkpoints! Name methods exactly as asked and instantiate objects in the given order.

Getting Started

- Run `cs0150_install lab2` to install this lab.
- Open up the `lab2` directory in Atom. You should see one stencil file, `App.java`.
- Let's make a class to represent a video player in a new file. Name it something appropriate - maybe `VideoPlayer`? This class should belong to the package `lab2` and **extend** `cs015.labs.VideoLabSupport.SupportVideoPlayer`. If you don't remember how to **extend** a class, revisit the lecture slides for more information.
- Write a blank constructor for your VideoPlayer class. This constructor will not need any parameters.
- In `App.java`, add a line in the `start` method underneath line 19. This added line should call `main.addVideoPlayer(...)` and pass that method a new instance of the `VideoPlayer` class.
- Try running the app at this point - in the console you'll see an error message alerting you to override the `addProviders` method. Don't worry - we will use this method later to allow the support code to display video.this.

Building a Display

- TV Content can come from many sources. Let's define one now.
- Create another class called `CartoonNetworkProvider` in a new file. Think of this class as representing a specific internet TV site, like HBO Go or Netflix. Remember to declare the package! Also don't forget a blank constructor here too. We will fill it in later!
- In this class, define a method that returns a `String`. Call this method `showAMovie`, and in the method, return the string `PopeyeForPres.mp4`. Our support code will find this video file and display it for you.
    - If you're not familiar with `Strings`, just think of them as another built-in Java type, like `int`. They can contain an arbitrary ordered sequence of alphanumeric characters, and are created by wrapping the contents of your string in quotes: `String dwightString = “Beets, Bears, Battlestar Galactica.”;`
- Oh no! We have no way of serving this video to the display! While we've written this method, no other part of our application knows about it. Remember the `addProviders` method we got an error about before? Let's take advantage of that now.

Checkpoint 1: Switch off with your partner here. While the other partner is coding, make sure you are assisting. In addition, begin to think about how an interface works and how it might be useful for this lab.

Adding a Provider

- In your `VideoPlayer.java` file, create an **instance** variable for your new `CartoonNetworkProvider`. Make sure to instantiate it in the constructor! Note that creating a new instance of one class within a different class counts as Containment!
- **Override** the `SupportVideoPlayer`'s abstract `addProviders` method in your `VideoPlayer`. If you don't remember how to **override** a method, revisit the lecture slides for more information. Refer to the code below to understand what you should put in your VideoPlayer file. **Note:** there is a bug in our code below! We have intentionally typed something incorrectly in order to give you the chance to start debugging your own programs!

```
@Override

public void addProviders(int x) {

      /* Your code here! */

}
```

- Try to compile your code. You should get the following error:

```
VideoPlayer.java:14: error: method does not override or implement a
method from a supertype

    @Override

    ^
```

  What does this mean? In our support code, the abstract `addProviders` method takes no parameters. But here, in your override, we've given you a method signature that takes in an **`int`**. When Java tries to compile this code, it considers such a method an invalid override, and prints out a helpful error for you.

- Fix the above method signature so that it is a valid override of our support method. *Hint:* we told you that our abstract method takes no parameters! When you've fixed the bug, and can compile without any errors, move on!
- We have provided a method within our **`SupportVideoPlayer`** called **`getChannelChooser`**. This will return an instance of a `ChannelChooser`, which only exists in support code but which manages registering different video providers.

- Fill in the overridden `addProviders` method by calling **`super.getChannelChooser`**, which returns an instance of ChannelChooser.
- Call `addProvider` on the instance of **`super.getChannelChooser`** and pass that method your `CartoonNetworkProvider`.
- Try running your app--look at the error message you get! As you can see, the app won't work with a specific type of video provider, because the video player has no way of knowing what functionality a specific provider has. This sounds like we need a way to give common functionality to a number of different video providers—how about an interface!

Welcome to Interfaces

Your First Interface

- Create a new interface in a new file. Call it what you like, but remember that good naming is important! Since this interface will model functionality for a number of different video providers, it might make sense to call it `VideoProvider`. Or `Provideable`. Just not `MandatoryLabInterface`.
  - If you need a refresher on how to create an interface, feel free to refer back to lecture slides or the lab reading.
- In this interface, declare two methods. Both will return `Strings`. One will return the name of the provider, and **must** be called...wait for it...**`getName`**. The other will return a `String` of the available videos for that provider, separated by commas, and **must** be called **`getTopVideos`**. Neither method needs any parameters.
- Modify your `CartoonNetwork` to implement your interface. Remember that you now need to implement every method defined in the interface. Do this now - remember that your **`getName`** method should probably return a String like **`“Cartoon Network Channel”`**.
  - Your **`getTopVideos`** method is a little more complicated. Your **`getTopVideos`** method must return a comma-separated list of the video file names available for that particular provider. We provide a list of the available videos on the following page. This might look weird! But eventually you'll end up with a string that looks similar to this: **`"PopeyeForPres.mp4,PopeyeCookingWithGags.mp4"`**, etc. where each video for that provider is listed with a comma (but no space) separating them.
- If you run the app now, you won't see an error message. Why is that? The reason is simple: now that you've implemented the `VideoProvider` interface, our support code knows that every video provider that you will write has those two methods implemented, which provides common functionality. It can now be *guaranteed* that any video provider passed to it will be able to return its name and its list of available videos. Cool, right?
- Run your video player and click on the button that reads “Cartoon Network Channel”. Now, the list of videos shows up along the bottom! Click on a video and watch it play.

Checkpoint 2: Check your interface with a TA!

Switch off with your partner here. As you assist your partner, think about how interfaces work with *more* classes implementing them. Discuss any benefits and downsides you see with this Object Oriented Design concept.

Showing Videos Generically

- Our video player is finally working! However, Dunder Mifflin employees have seen plenty of old cartoons, and they'd like to be able to access content from more sources. Let's define a couple more channels.
- Create two new classes, one to display cute animal videos (Nature, maybe?) and one to display video game videos (Twitch TV?). Both should implement your interface and its methods. Remember to instantiate them in the constructor of your **`VideoPlayer.java`** and store them as instance variables. Also make sure to repeat your procedure in **`addProviders`** in **`VideoPlayer.java`** to add these classes to the display. The names of the videos to provide in the **`getTopVideos`** method are given below. These classes should look similar to your **`CartoonNetworkProvider`** class! If you get stuck, look back at your earlier work.
- Run your app. You should be able to see the three channels and play videos from each.

List of Available Videos

These are the names of the video files you should return in the **`getTopVideos`** method of each class that implements your interface. Remember that you should be returning a comma-separated list of these video names.

Cartoon Network Channel

- `PopeyeCookingWithGags.mp4`
- `PopeyeForPres.mp4`
- `PopeyeMeetsSinbad.mp4`

Nature Channel

- `Elephant.mp4`
- `Koala.mp4`
- `Turtle.mp4`

Twitch TV Channel

- `ChronoTrigger.mp4`
- `SuperMarioBros.mp4`
- `PokemonRed.mp4`

Summary

The purpose of this lab was to teach you how to implement polymorphism using an interface and inheritance. The idea behind polymorphism is that a single instance of an object can be of different types. Originally, the parameter passed to the **`VideoPlayer`**`'s` **`addProvider(...)`** method was a **`CartoonNetworkProvider`**. By adding the interface, the object was not only of type **`CartoonNetworkProvider`**, but also of type **`VideoProvider`**, a much more generic type that can be common for a lot of **`VideoPlayers`**, such as HBO, Netflix, and Hulu. The **`VideoProvider`** interface factors out video-playing commonality. This is a widely used design pattern to write more generic code, allowing the programmer to write one method instead of many.

Some final remarks about interfaces:

Interfaces can also extend from other interfaces. For example, if you had a **`Singable`** interface that required a **`sing()`** method, you could extend this interface to create a **`Musical`** interface. It could look as follows:

```
public interface Musical extends Singable {
      public void act();
      public void beDramatic();
}
```

Anything that implemented the **`Musical`** interface would have to define all the methods in **`Musical`** as well as all the methods in **`Singable`**.

Don't forget, a class can implement multiple interfaces. The **`CartoonNetworkProvider`** class you wrote could also implement a **`AudioProvider`** interface, as well as any other you deem necessary. You would separate each additional interface with a comma. The syntax to do that would be:

```
public class CartoonNetworkProvider implements VideoProvider, AudioProvider {
    /* Code here. */
}
```

Checkpoint 3: Check your **`VideoPlayer`** program with a TA!