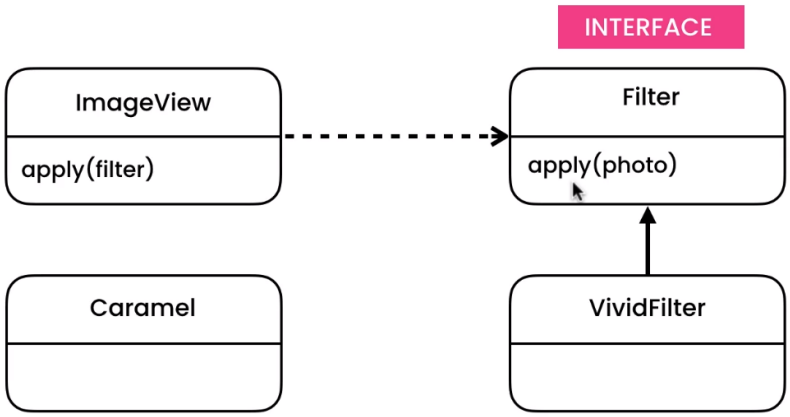
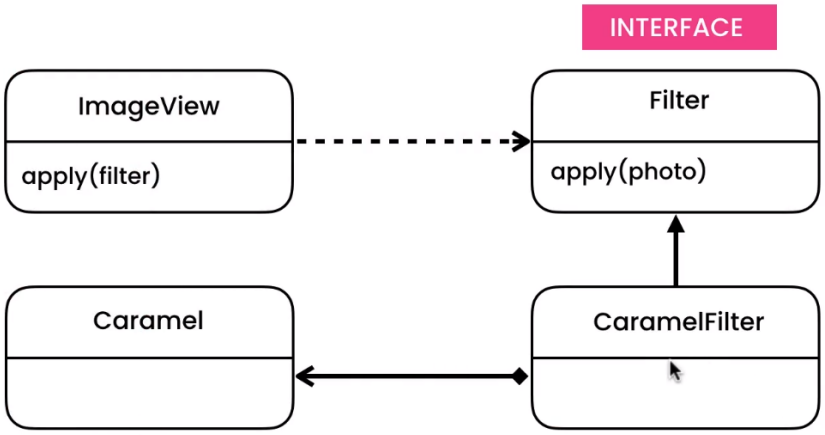
Adapter Pattern

Example Problem - You are developing a mobile app that provides image editing capabilities, Eacvh image class can have various filters applied to it, with each filter implementing the Filter interface. Thius the apply method of the ImageView class can expect a class which implements this filter inteface. However you wish to make use of a third party libaray of filters, but none of these implment the filter interface causing the apply method to break. In order to fix this we can "adapt" these classes in the third party filter library to work with our ImageView class using the adapter method. As below we can't use the Caramel class below as it does not implement the filter interface :



We can adapt the interface of this caramel class by introducing a caramel filter which is composed of the Caramael class :



CaramelFilter is the Adapter and Caramel is the Adaptee. We could also extend teh Caramel class in the Adapter class, but we favour Compostion over Inheritance as you can only extend one class and if the filter interface was an abstract class then we could not inherit the Caramel clas as well in the Adapter class.


```
public class ImageView {

  private Image image;

 public ImageView(Image image) {
    this.image = image;
 }

  public void apply(Filter filter) {
    filter.apply(image);
 }
}
```

```
public class Image {
}
```

```
public interface Filter {
  void apply(Image image);
}
```

```
public class VividFilter implements Filter {
  @Override
  public void apply(Image image) {
    System.out.println("Applying Vivid Filter");
  }
}
```

From Thir Party Library			

`public class Caramel {`
`  public void init() {`
 	
`  }`
 	
`  public void render(Image image) {`
`    System.out.println("Applying Caramel Filter");`
`  }`
`}`
	`public class CaramelAdapter extends Caramel implements Filter {`
	`  @Override`
	`  public void apply(Image image) {`
	`    init();`
	`  render(image);`
	`  }`
	`}`
		`public class CaramelFilter implements Filter {`
		
		`//Composition favored over inheritance`
		
		`  private Caramel caramel;`
		
		`  public CaramelFilter(Caramel caramel) {`
		`    this.caramel = caramel;`
		`  }`
		
		`  @Override`
		`  public void apply(Image image) {`
		`    caramel.init();`
		`  caramel.render(image);`
		`  }`
		`}`
			`public class Main {`
			
			`public static void main(String[] args) {`
			
			var imageView = new ImageView(new Image());
			
			//throws compilation error
			
			imageView.apply(new Caramel());
			
			//works because CaramelAdapter implements Filter interface or you could use the CarmelFIlter
			
			imageView.apply(new CaramelAdapter());
			
			}
			
			}