Feature 1: Resource File Processing

As Glide is an open source media management and image loading framework, resource processing is definitely an essential feature. In assignment 1, we examined the features of the systems within a sample module that displays and processes svg images. However, since it is more meaningful to look at how this feature works across all types of media resources, we will reexamine this feature at a higher level.

The Resource File Processing feature can be broken down into three components: Encoding, Decoding, and Transcoding. These are important components because they allow users to transition between image file types smoothly. An important characteristic we have observed of these three interfaces is that they are adaptable, allowing the implementation of more specific encoders, decoders, and transcoders to address varying file types.

Encoder

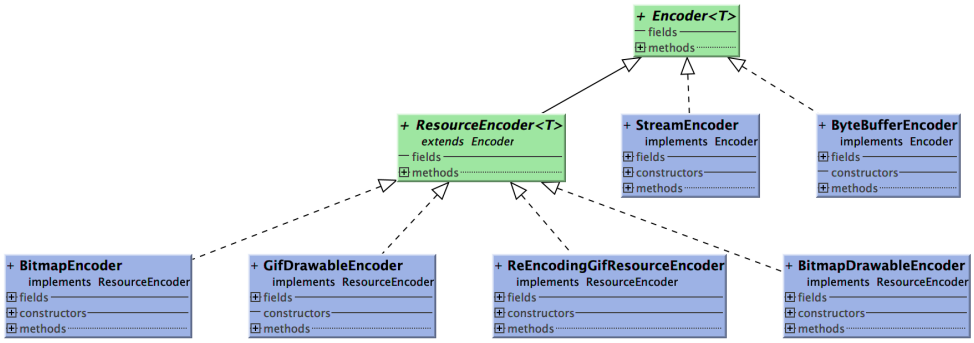
Encoder.java is located in the file path */library/src/main/java/com/bumptech/glide/load/.* It is an interface for writing data to some persistent data store such as a local File cache. Its encode() method writes data to the given output type and returns a boolean to signify whether the data has been written successfully and should be committed.

```java
public interface Encoder<T> {
  /**
   * Writes the given data to the given output stream and returns True if the write completed
   * successfully and should be committed.
   *
   * @param data The data to write.
   * @param file The File to write the data to.
   * @param options The put of options to apply when encoding.
   */
  boolean encode(@NonNull T data, @NonNull File file, @NonNull Options options);
}
```

/library/src/main/java/com/bumptech/glide/load/ResourceEncoder.java extends *Encoder* class, also defining a new method *getEncodeStrategy()* which will return an *EncodeStrategy*, an enum class, which details how a *ResourceEncoder* will encode a resource to cache.

```java
/**
 * An interface for writing data from a resource to some persistent data store (i.e. a local File
 * cache).
 *
 * @param <T> The type of the data contained by the resource.
 */
public interface ResourceEncoder<T> extends Encoder<Resource<T>> {
  // specializing the generic arguments
  @NonNull
  EncodeStrategy getEncodeStrategy(@NonNull Options options);
}
```

There are multiple classes that implement both the *Encoder* and *ResourceEncoder* interfaces. This allows the system to account for a range of file types and leaves the possibility for further implementations.



Decoder

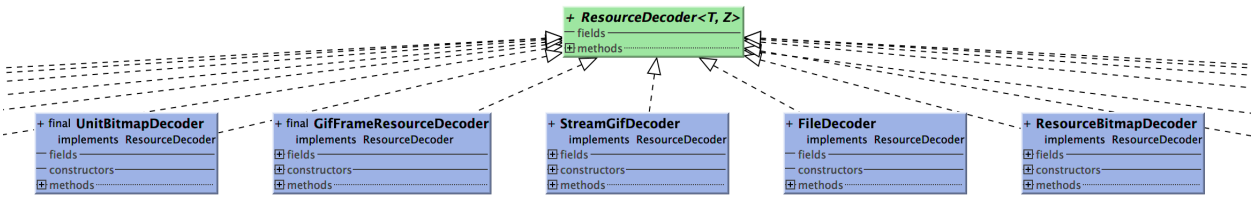
ResourceDecoder.java is located in the path: */library/src/main/java/com/bumptech/glideload/*. It is an interface for decoding resources. It takes in parameters T as shown in figure below and returns a decoded resource Z.

```
/**
 * An interface for decoding resources.
 *
 * @param <T> The type the resource will be decoded from (File, InputStream etc).
 * @param <Z> The type of the decoded resource (Bitmap, Drawable etc).
 */
public interface ResourceDecoder<T, Z> {
```

Similar to the *Encoder.java* interface, there are many implementations of *ResourceDecoder* to capture many of the image file types that may need to be decoded. They all override the handler() method, which returns a boolean indicating whether resource T is able to decode, and the decode() method, which decodes resource T as the format of ideal width, height, and expected type.

```
@Nullable
Resource<Z> decode(@NonNull T source, int width, int height, @NonNull Options options)
    throws IOException;
```

Note on the image below the many relationship arrows pointing to ResourceDecoder, which are all implementations using different data types.



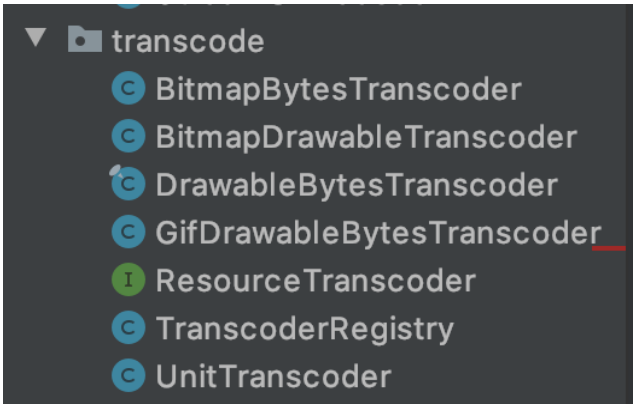
Transcoder

ResourceTranscoder.java is located in the path: */library/src/main/java/com/bumptech/glide/load/resource/transcode.* It is an interface used to transcode resources. This allows the system to alter a given resource which is of type Z into type R, which is a compatible type for Android. It has one method, transcode(), which takes in a resource type to convert as expected type Options.

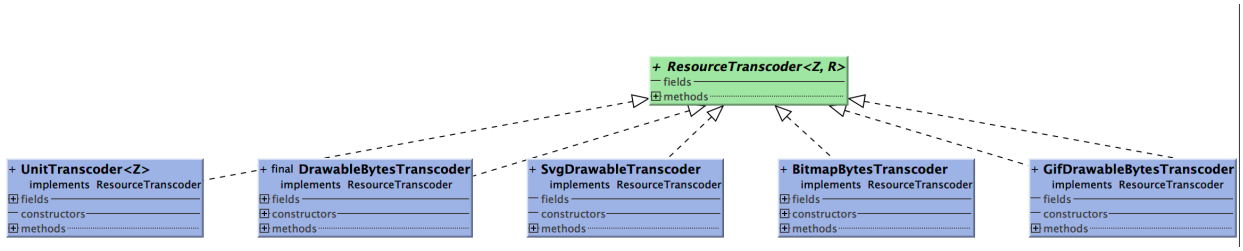
```
public interface ResourceTranscoder<Z, R> {

  /**
   * Transcodes the given resource to the new resource type and returns the new resource.
   *
   * @param toTranscode The resource to transcode.
   */
  @Nullable
  Resource<R> transcode(@NonNull Resource<Z> toTranscode, @NonNull Options options);
}
```

The */library/src/main/java/com/bumptech/glide/load/resource/transcode/* folder contains all kinds of transcoder classes, which implement the ResourceTranscoder interface.



These different implementations allow the user to convert from a range of image file types. We can see this relationship on the UML diagram below:



Other relevant parts of the system and how they're relevant:
This feature is highly relevant for another key feature, which is to display media resources such as images, video stills, and GIFs. The file processing feature essentially is the first step that handles resources as inputs, which will be output as display.