Rationale for Resubmission:
Our initial report format was closer to that of a documentation. We included UML diagrams without necessarily referring to them or describing what is being presented. Further, we included a lot of code snippets that do not enhance the understanding of the overall functionality of each feature. Lastly, our original report lacked structure. Below is an index of the changes we made to fill the deficit in our original report:

1. Restructured our report into three parts; Feature Overview, Relevant Components, and Component Behaviors
2. Instead of using code snippets to show relevant classes and what methods are called in each class, we used abstractions to describe the behaviors of each component.
3. We elaborate on the relationship of the components to other classes in the system through the use of UML diagrams, referring specifically to each relationship discussed.
4. Added figure labels and created a section for all figures mentioned in the report.

Resource File Processing Feature

Feature Overview

Glide is an open source media management and image loading framework, which implies it must somehow process existing or new file resources in order to load, display, and cache them. Since file processing is a critical first step that enables the overall functionality of the system, we decided to expand on our initial examination of the feature from assignment 1.

The resource file processing feature can be broken down into three essential components; Encoder, ResourceDecoder, and ResourceTranscoder. An important characteristic we have observed of these three components are that they are adaptable interfaces that allow the implementation of the file processing feature to address varying file types.

Relevant Components

The three interface components that are related to the feature are Encoder, ResourceDecoder, and ResourceTranscoder. Encoder (/library/src/main/java/com/bumptech/glide/load/) is an interface that writes data to a persistent data store such as a local file cache. ResourceDecoder (/library/src/main/java/com/bumptech/glide/load/) is an interface that decodes resources. It takes the file structure the resource will be decoded from, such as a File or an InputStream, and returns the type of the decoded resource, such as Bimap or Drawable. Lastly, ResourceTranscoder (library/src/main/java/com/bumptech/glide/load/resource/transcode) is an interface used to transcode the resources to allow the system to alter a given resource to an Android-compatible file type.

Component Behaviors

Since Encoder, ResourceDecoder, and ResourceTranscoder are interfaces, it is useful to examine how they are implemented in the overall system using a UML diagram, then look at how the relationships fit into the overall system feature.

The Encoder uses an encode() method to write data to the given output type, such as a File or Stream, then returns a boolean to signify whether the data has been written successfully. Figure 1 shows that the Encoder interface is extended by ResourceEncoder (library/src/main/java/com/bumptech/glide/load/), which defines a new method getEncodeStrategy(), which returns an enum class, EncodeStrategy, that details how a ResourceEncoder will encode a resource to cache. The ResourceEncoder is implemented by four classes, BitmapEncoder, GifDrawableEncoder, ReEncodingGifResourceEncoder, and BitmapDrawableEncoder. Each of the classes implement the ResourceEncoder interface that passes a specified output type to write data into and they call the getEncodeStrategy() and encode() methods write the original bytes of the file type into the output type.

The ResourceDecoder contains a handles() method that returns a boolean that indicates whether the decoder is capable of decoding the given resource with the given options, and a decode() method that returns a decoded resource from the given data if a resource could be decoded. Figure 2 illustrates that the ResourceDecoder interface is implemented by a number of classes; each of which address a specific data type that must be handled and decoded. The FileDecoder class, for example, implements ResourceDecoder to create a resource for a given File resource type.

Lastly, the ResourceTranscoder implements a transcode() method that takes in a given resource to convert into the new resource type and returns the new resource. Figure 3 demonstrates a relationship between a series of classes that address varying data types to transcode. SvgDrawableResourceTranscoder class, for example, implements ResourceResourceTranscoder to convert the SVG's internal representation to a Picture resource that is Android-compatible.

As such, the Encoder, ResourceDecoder, and ResourceTranscoder are three major components that complete the process of encoding a resource into a file or stream, decoding them and making sure such data type is decodable, and lastly converting them into a specific file type that is Android-compatible. 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.

Figures

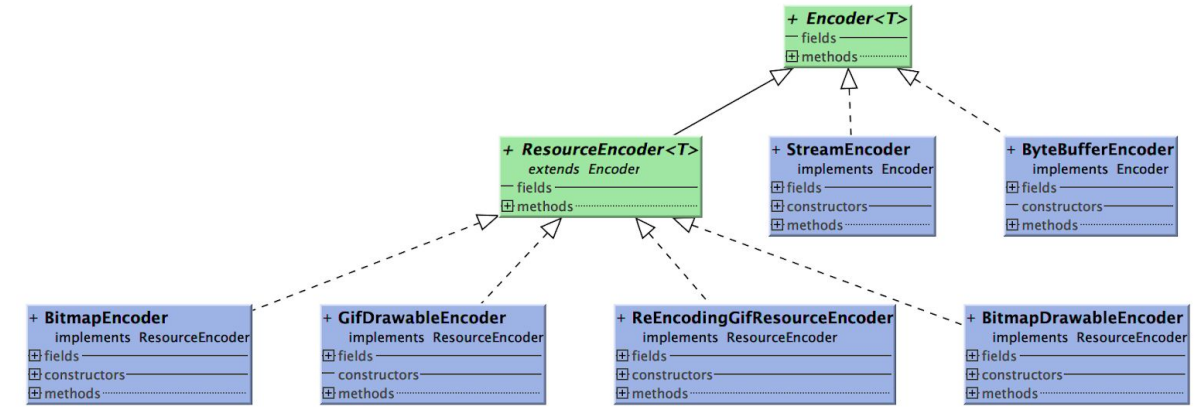

Figure 1: ResourceEncoder and Encoder Interfaces and their Implementation in the System

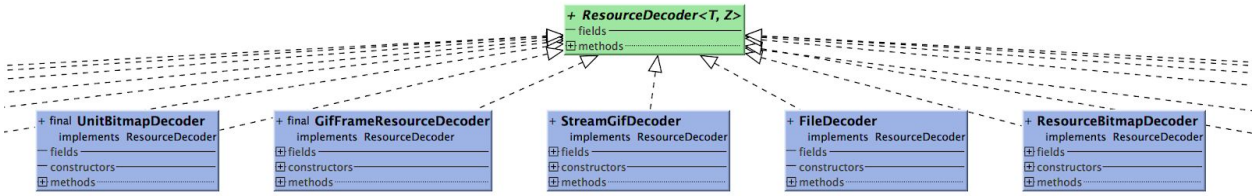

Figure 2: ResourceDecoder Class and its Functional Implementation in the System

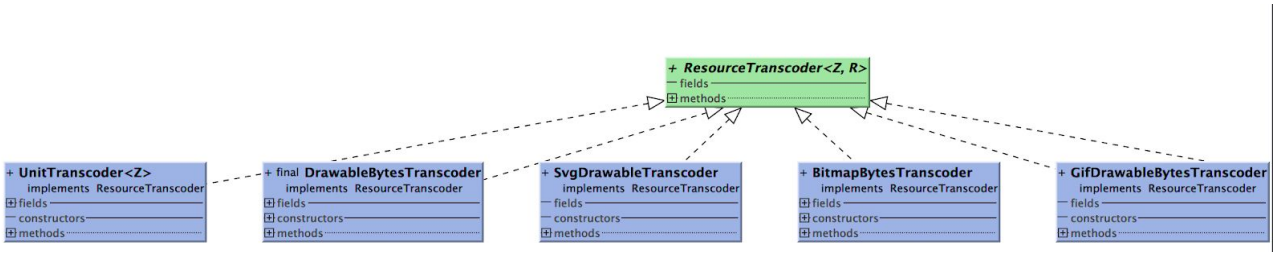

Figure 3: ResourceTranscoder Class and its Functional Implementation in the System