Homework 4

Team Costco | Marc Andrada, Soobin Choi, Duo Chai

Rationale for Resubmission:

Given key feedback from our TA we have approached this assignment in a more thorough and organized manner. We have made changes to our assignment in the following ways:

- Updated our report structure to be cleaner/more organized to allow better flow of ideas and information.
- Adjusted our architectural model to not forcibly constrain itself to the MVC style but instead exhibit a more layered architectural style.
- Cleaned up Issues and Pull Request section to be more report-like instead of just a listing of information.

Updated report can be found starting on the next page.

Glide

Architecture | Social Context | Pull Requests | Issues

Introduction

We will first examine Glide's as-implemented architectural structure. Then we will display information regarding the social context of this system. Finally we will explore several interesting pull requests and issues.

Glide Architecture

Determining the Architecture

The landing page for documentation related to the Glide system is on github (https://bumptech.github.io/glide/). The website provides documentation for Glide components that can be aggregated into Glide's major features such as efficient image loading, processing, and caching. Since the system architecture is not explicitly discussed, we attempt to document and produce the system architecture based on available resources such as this documentation, system source code, and the UML diagram.

In class, we applied a bottom-up comprehension method to derive the architecture diagram for JPacman4. Since the Glide system is much larger, such comprehension strategy poses various challenges; so instead, we recovered the architecture of Glide through top-down comprehension of the system. We took a look at Glide's website and used the documentation to extract meaningful features of the system at high-level. We then tried to determine the relationships between the behaviors among these meaningful features by referring to the source code and folder hierarchies. We have come to the conclusion that our system resembles a layered architecture. We first provide a macro overview image of our architecture including the user's view for visualization in Figure 1.1 below. A more detailed breakdown of this model will follow.

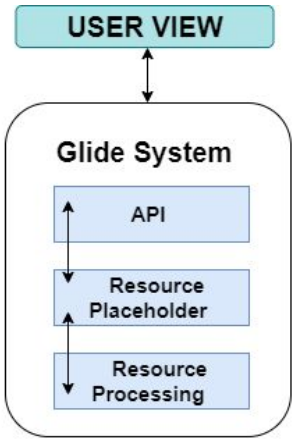

Figure 1.1 - Macro View of System Architecture

Our initial understanding of Glide is that it is a library that provides a smooth way for applications to display images. With this in mind we first thought to model our architecture after the MVC (Model, View, Controller) style. However, upon further consideration of our system we realized that it did not explicitly contain the three MVC components. Instead, the user's "view" can be seen as external to our system. This view is bridged to Glide's API by accessing the library via its method requests and responses. The lower levels of our system are separated into two parts: the placeholding of resources to be displayed and the processing of these resources. Holistically speaking, the user's view is connected to the actual system and its inner logic via API methods. The system itself is therefore more appropriately observed as a layered architecture. This is modeled in Figure 1.2 below.

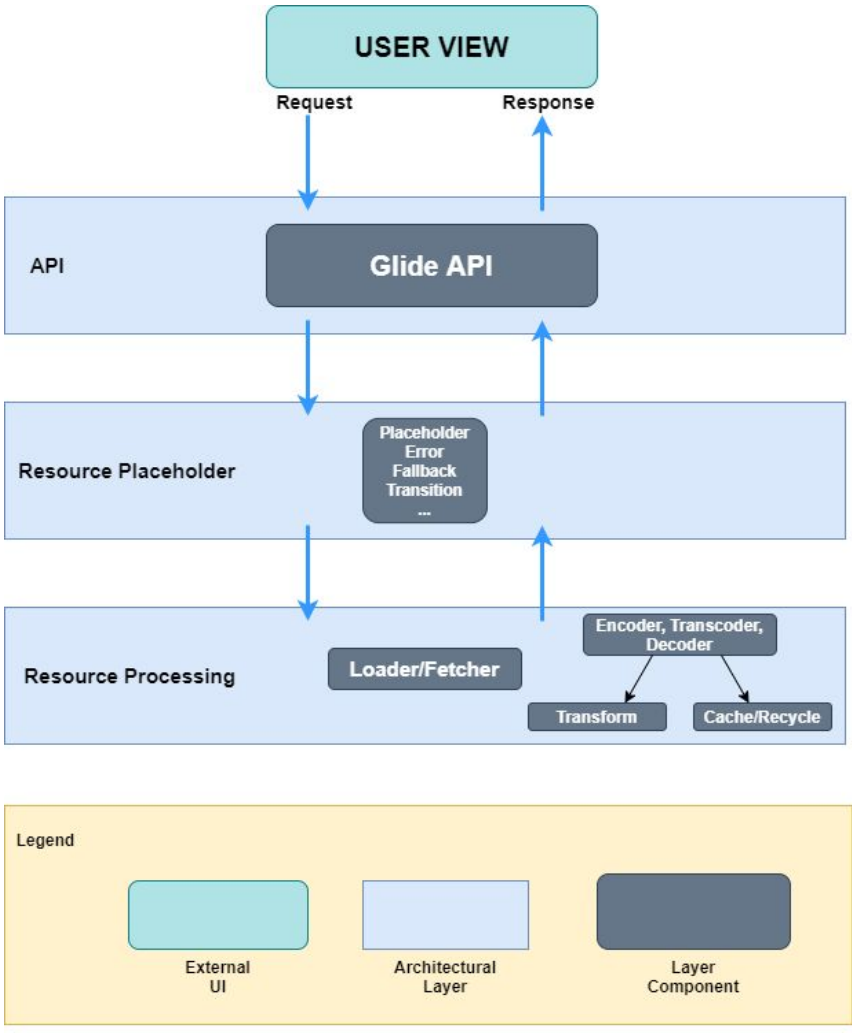

Figure 1.2 - Detailed Layered Architecture

Processing Layer

Overview

When creating a Glide request, we are specifically dealing with resources of different data types (such as bitmap, bytes, drawable, file, gif, etc.). The operations that deal with these resources, which we interpret as “Processing” are stored in the Glide/src/library/load/resource/ folder. This layer interacts with the placeholder layer, where the placeholder requests the resources needed to be processed in order to be loaded and displayed properly. The components of this layer are described below starting with the lowest layer and moving up.

Fetcher/Loader/Processing

The Fetcher component (implemented in Glide/src/library/load/data/ folder) fetches data from URI pointing to a local resource. The Loader component(implemented in Glide/src/library/load/model/ folder) translates a specific model into a data type that can be decoded into a resource. Then, through a series of resource processing operations, such as encoding, decoding, and transcoding, the system makes a resource of any data type available for building a request.

Caching

In addition to processing resources, the system also caches resource data. We observed a bidirectional relationship between resources that are processed and operations that generate cache keys from original resource data (can be either remote or local) to either load data from cache or skip caching based on the caching strategy. Caching and recycling of the program provides the ability for Glide to retrieve resources faster and more efficiently by checking different layers of caches before starting a new request for a resource such as image. The classes and interfaces related to caching and recycling are stored in the Glide/src/library/load/engine/ folder.

Transforming

The transformation component also maintains a bidirectional relationship with resources. The operation will take a resource, mutate it if possible, and return the mutated (or un-mutated) resource, which can also be used to create cache keys through the system’s cachers and recyclers. The classes related to resource transformations are accessible throughout the Glide/src/library/load/ folder.

Placeholder Layer

Overview

The “Placeholder”, which we have defined in our architecture, provides different methods to external applications to control how the different types of resources are being loaded and displayed into their View object. This layer interacts with both the processing layer and the API layer. We dove deep into the Glide/src/library/ folder. Glide/src/library/request/target/ folder stores different target options, which are responsible for both displaying placeholders and loaded resources and determining the appropriate dimensions to ensure that images are loaded properly into the views for each request. The components that affect how “Placeholders” behave is through the way transitions are defined within the library.

Transition

Transition is a controller object that exists between the user interface of the external application and the resources that are processed through Glide. Glide/src/library/request/transition/ folder stores different transition options. The different transition options, which can be adjusted in transition factories, determine how a drawable transitions from a placeholder into a new image view.

Error/Fallback

The remaining files in Glide/src/library/request/ folder and the methods defined in Glide/src/library/ folders contribute to the implementation of other Placeholders, or Drawables that are shown while a request is in progress, such as an error, fallback, and placeholder drawables.

API Layer

Overview

The Glide API layer facilitates the communication between the user view and the placeholder layer of our system. The user view is represented as external applications interacting with the API.

Glide API

Depending on the requirements of the developers that are adopting Glide, the library can be utilized to fulfill specific and customized functionality for their system. Reliable and effective communication between these two components is essential for Glide to achieve it’s end goal of providing a “flexible API”, which allows:

1) Scrolling any kind of a list of images to be as smooth and fast as possible
2) Effective fetching, displaying, caching of both remote and local images
3) Capabilities for plugging it in to almost any network stack.

Social Context

Project State

Overview

Glide is still actively under development with both contributors and maintainers more recently focusing on improvements to overall performance of the system and fixing particular bugs that arise. The latest release date was on Jan 8, 2020. And the latest PR was merged on Feb 25, 2020. There have been 2472 commits as of Feb 25, 2020. Observing from the commit history, there are a few new PRs being merged weekly. There have also been 144 open issues and 3484 closed issues as of Feb 25, 2020. We observed that maintainers deal with and close open issues once a week. Sjudd appears to be the only active maintainer for now. But there are many developers, since a lot of people contribute by creating new PRs: mkj-gram, ouattararomuald, landicefu, and etc.

Relevance

Glide's attains a decently high level of usage. It has been forked by 5.2k, watched by 1.1k and starred by 28.5k users compared to Picasso, a similar image loading/displaying library, which has 4k, 921, 17.3k respectively.

Standards

GitHub

PRs must include a description, motivation and context; however, they don't have a template established for issues (Figure 1.3)

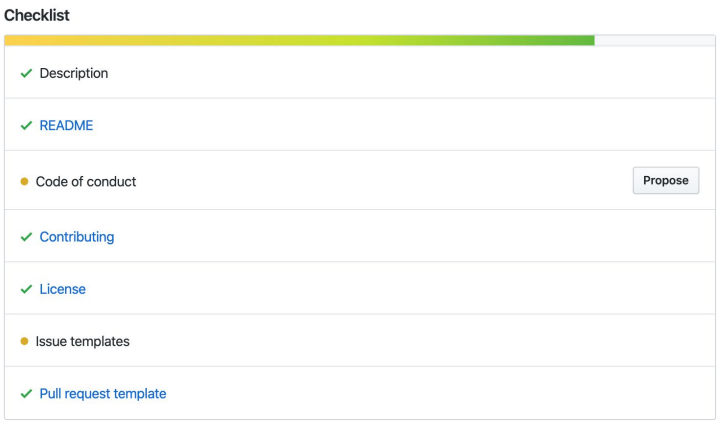

Figure 1.3 - Checklist of GitHub contribution standards

Contribution Process

Overview

According to https://github.com/bumptech/glide/releases releases do not follow a development roadmap that are based on system updates determined by Glide's development team. Instead, they seem to push releases twice a year after enough commits have been added to the system. Glide evolves through weekly PR merges, observing from their activities on merging PR and closing issues.

In general, all contributions are welcome as mentioned here (https://github.com/bumptech/glide/blob/master/CONTRIBUTING.md). Since Github is the only way to contribute code changes, contributors must sign the Google's individual contributor license agreement before contributing. Although they have a defined code style in their IntelliJ code files in the repo, contributors who aren't able to pass the code styles are encouraged to submit a pull request anyway as the team will provide support where necessary.

The lifecycle of submitting codes is not exactly the same as what we discussed in class since they won't assign a developer to fix a problem and will only upgrade Glide by merging PRs. By observing the lifecycle of how maintainers merge PRs, we discovered that they practice the following:

- Find a bug/improvement
- Create a new PR describing the problem, motivation, and context
- Test the PRs if maintainers think it is meaningful and the method used to solve the problem can be maintained from a long term perspective
- Merge PRs that pass the test, otherwise close the PR.

Support Tools

Issue/PR Bots

By observing the PR lifecycle, we found they use Bots such as googlebot, stalebot (Automatically add CLA tags and close stale Issues and Pull Requests that tend to accumulate during a project).

Version Control

Since they use github as the main platform to share debugging and code changes for all aspects of libraries, we can assume they also use version control to maintain the system on their end.

Build/Integration Tools

- TravisCI: To test the project build.
- Checkstyle: To check coding styles.
- Maven central: To check Maven dependencies in the local repository, and will search in the Maven central repository if dependency cannot be found locally.

Pull Requests

Update ByteBufferStream read() to return byte values from 0-255 or -1 if EOF reached

Source: https://github.com/bumptech/glide/pull/3887

Adjusted ByteBufferStream read(), by adding a bitmap mask ‘0xff’. After masking by ‘& 0xff’, it will leave the value in the last 8 bits of the bit stream, promising the accuracy of bit computing.

Add a VideoDecoder for ByteBuffers

Source: https://github.com/bumptech/glide/pull/4033

Resolved an issue with VideoDecoder, where if the given DiskCacheStrategy: RESOURCE or NONE, video files would not be properly read and converted by the decoder. The pull request appears to resolve this by implementing a byte buffer to the VideoDecoder and was merged. This pull request also directly addresses an issue opened earlier here.

Improve handling of EOF in DefaultImageHeaderParser.Reader

Source: https://github.com/bumptech/glide/pull/3952/files

Original code in DefalutImageHeaderParser.java does not support EOF exception handling. The PR handles additional failures on any that reads or parses byte array data. Seems like an oversight at the time of initial development since the system deals with various data types and error handling is an important aspect.

Update targetSdkVersion to 28

Source: https://github.com/bumptech/glide/pull/3880

Updated SDK to meet with Google Play’s API requirements. The conversion didn’t affect the build of the system per the Travis CI tests. This PR is interesting because it shows possible best practices when it comes to keeping a system up to date. The developer seems to be considering the future of the system by updating to the new SDK.

Improve comments and consistency of sampling in DownsampleStrategies

Source: https://github.com/bumptech/glide/pull/3703/files

Good comments in a large system like Glide will help users tremendously in regards to what components play a role and how they are related. This PR not only improves downsampling the dimensions for image resources but also add helpful comments detailing how the downsample strategy changes when the system is developed and run on KitKat API.

Issues

Glide.with(Context) very very slow (takes 1 sec)

Source: https://github.com/bumptech/glide/issues/3876

This is an issue with performance. Apparently, the initialization of Glide takes “too long” (1 second). One of the maintainers mentioned that they recognized this issue and proposed a soft solution of allowing glide to be “lazily initialized”. By this, we believe the maintainer means to initialize the Glide class as close as possible to when it will be utilized for example when loading an image. Other comments appear interested in a more formal solution to addressing the performance.

Certain GIFs display red rectangles after 3.7

Source: https://github.com/bumptech/glide/issues/3319

Issues with certain gif files after 3.7 update. “Red rectangles” appear in the frame of the loaded gif. There rises the question of whether it is just bad files or something with how the gif file is being loaded using glide. Currently no PR has been attempted, but ideas are being sought after.

In a weak network (3G), the local picture is displayed only after the network picture fails to load.

Source: https://github.com/bumptech/glide/issues/2998

There appears to be a lot of discussion on this issue, which is why we wanted to look into it further. The main issue appears to be difficulty loading an image depending on the type of network utilized. The networks considered were GSM and UMTS. These are types that can be determined in the android emulator settings. A potential workaround for this has been proposed, which would involve “reading image bytes into memory and wrap it with the ByteArrayInputStream”. The user who proposed this noted that it would take away from a currently standing method in the system: setPriority.

Glide's `trimMemory()` implementation doesn't trim when foregrounded

Source: https://github.com/bumptech/glide/issues/2810

This issue revolves around problems with memory “trimming” and management depending on whether the application is in the foreground or background. The outside developer noted that when the application is in the foreground, a call to Glide’s method: `android.content.ComponentCallbacks2.TRIM_MEMORY_MODERATE` does not return a high enough

integer to trim memory. On the other hand, when the application is in the background, the same call returns a value large enough to trigger memory trimming.

[Question] How to prevent clearing an ImageView before the load is complete?

Source: https://github.com/bumptech/glide/issues/2505

The issue is mainly requesting help with locating useful features within current documentation involving their concern with image load transitions. A user was wondering whether they could prevent Glide from clearing their image from an image view before loading is complete. Everytime he refreshed the image, it would flicker white for a second before it was completely loaded, unless it was currently present in the memory cache. The user was looking for updated documentation for a method called dontAnimate() hoping this could disable the transition issue he was having. According to the maintainers, this transition is necessary to prevent potential memory leaks from occurring due to frequent refresh calls. They suggested utilizing a ViewSwitcher class, which can display an image and wait for the next to load before transitioning.