SoftPhone

Milestone 3: Code Smells and Refactoring

Submitted By:

Vishal Dhaduk (6714552)

Harshit Shah (6557198)

Rahulkumar Patel (6557171)

Jay Thakkar (6557163)

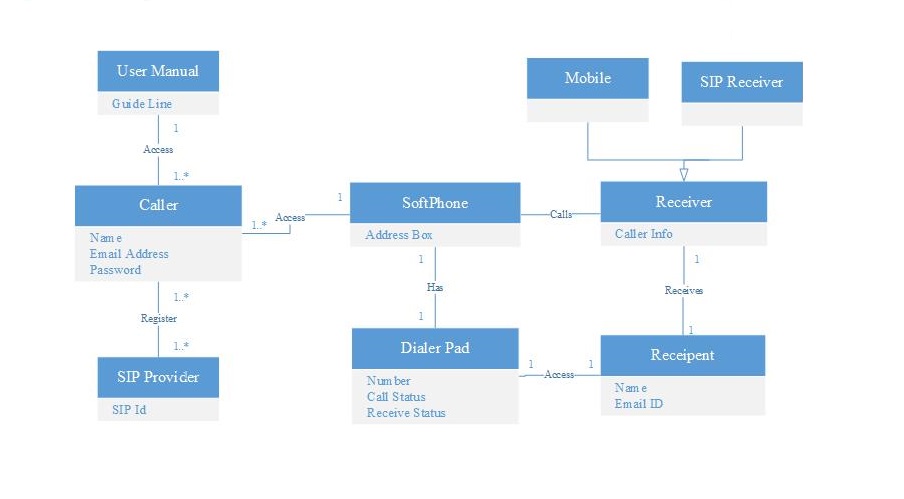
Harsh Shah (6715591)

**GitHub Repository:** <https://github.com/hash19/SoftPhone>

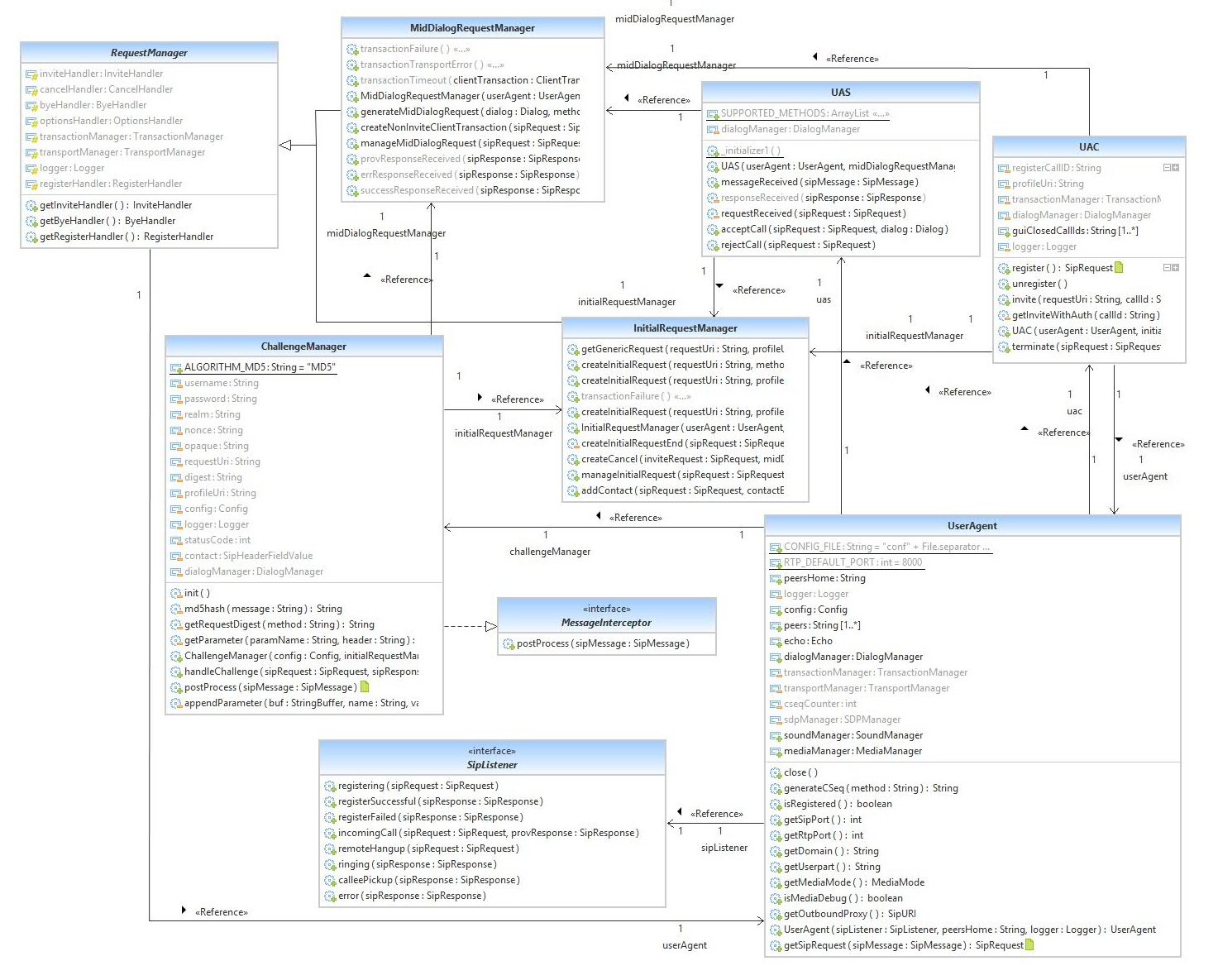
# Summary of Project

[Peers Java SIP SoftPhone](http://peers.sourceforge.net/) is a very simple telephony application. This application work as a Session Initiation Protocol (SIP) User Agent on client side. It handles all Client side responsibilities for establishing SIP based communication. SIP is Internet Engineering Task Force (IETF) protocol for VoIP. SIP phones can be implemented in a hardware or software form. As the name of this application suggest this is software based SIP phone. This application is developed in wholly in Java. It works well in windows, Linux and Mac operating system. It is required to have a SIP account to use this software. Three parameters named Username, Domain Name and password are needed to place phone calls.

# Conceptual Domain Diagram



# Class Diagram of Actual System



There are around ten packages in the source code. Although User-agent and User-agent handler package in source code is core layer or we can say brain of SIP softPhone. Our main focus is on these two packages.

Here, there are two types of classes in the User - agent package, Request Managers and Handlers. InitialRequestManager class is responsible for initial request specific handling and MiddialogRequestManager class is responsible for subsequent request specific handling. Handler classes implement method specific code. To support complexity each single method feature implemented in specific handler. Some of the example classes of these handlers are RegisterHandler, InviteHandler, CancelHandler etc. User-agent contains both User-agent client(UAC) and User-agent server(UAS). User-agent client is responsible for sending request and a User-agent client is responsible for incoming requests processing.

SipLisstner is being used as an interface for communication between user-agent(core layer) and graphical user interface(GUI layer). So, this separation can be identified easily. This SipListner provides user interface various SIP events such as incoming call, call pickup, call reject. UAC and UAS classes provides methods for these events. These methods are named as acceptCall(), rejectCall() etc. This is very good design in terms of software architecture. As Siplistner allows gui independent communication.There is a generic way of communication between core layer and user interface layer.

So, compare to domain diagram, user agent package contains classes which are similar to caller, softPhone and receiver class of domain diagram in terms of providing functionality. Although user agent package is independent of the receiver and mainly focus on the SIP part of the softPhone.

# C:\Users\JAY\Downloads\class relationship (1).png

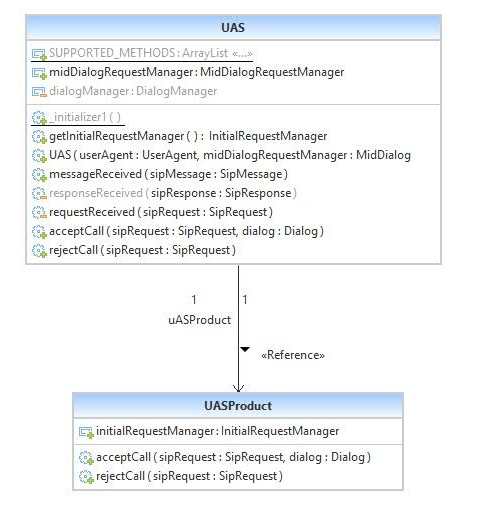
Here, Instances of InitialRequestManager class are referenced in UAC and UAS. InitialRequestManager class creates requets. Here user-agent Client (UAC) contains methods to register, unregister,invite etc. user-agent Server (UAS) contains methods for accepting calls, reject calls. Both UAC and UAS have method for getting an instance of InitialRequestManager class. The above given class diagram clearly shows the relationship among these three classes.

# 

# Code Smells and System Level Refactorings

* **God Class**

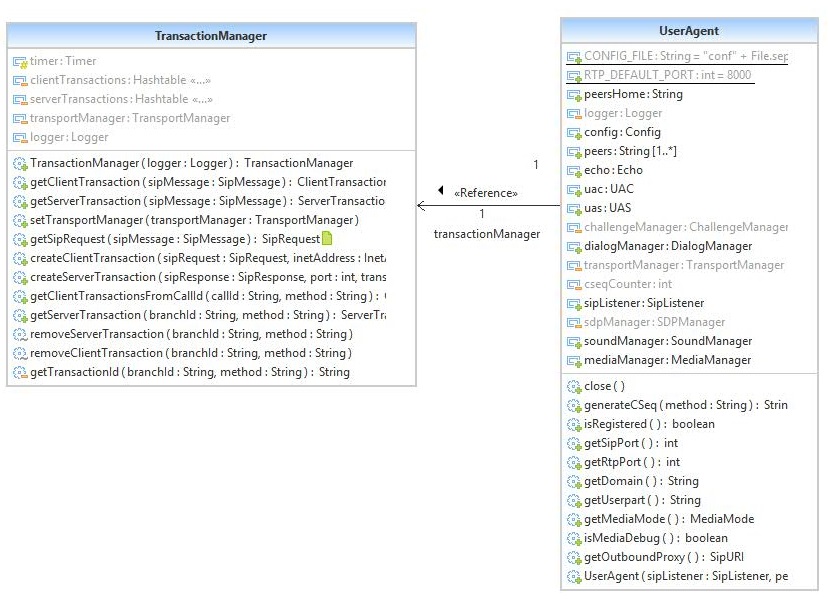
If the class has too many things to do then it is one of the cause of existence of duplicate code. God class consist of many variable with similar name or functionalities. In the god class there are too many responsibilities to do. As per the Single Responsibility Responsibility, A class should have one responsibility to do. God class violates this rule. Also the god class may have a very high cohesion. It is good to have a high cohesion, but very high cohesion makes the class too complex. This reduces the readability, analyzability and maintainability of class. To overcome this problem, we may create the extract class by subtracting the common variable in that class. By creating the extract class we can reduce the cohesion and responsibility of one class.



In the SoftPhone, UAS.java is god class. We can create the UASProduct.java extract class by extracting the UASProduct(), rejectCall (SipRequest) etc. method. By creating the extracted class we can reduce the duplication of code and responsibility of UAS.java Class.

* **Feature Envy**

Another code smell we found in source code is Feature Envy. There are a few classes in the user agent package that use methods of other classes excessively. This code smell increases coupling among classes. It also reduces maintainability and modifiability. Again, refactoring we can do in this case is extract method or move method. Both of these techniques are related to system level refactoring. Application of the above mentioned techniques may affect the architecture of the system.



* **Long Method**

Long method is one of the most common code smell in the source code of SoftPhone.It is known that the longer the method is, the more difficult to understand it. This bad smell often gives hard time to programmer to maintain it. Furthermore, bug featured in a long method is hard to detect and very hard to fix. It is always advisable that method should always focus on one task. A method with lots of parameters can also be called as Long method. We can replace that method by creating a method object. Here in the softPhone, *InitialRequestManager.java* class consists long methods. In *SipRequest createInitialRequest* method of that class contains lots of parameters. For extract this method, we can replace by creating objects of messageintercepter.

# Specific Refactorings that will be implemented

In above section of this document we have mentioned some bad code smells that we think exists in the UserAgent package of the source code. These bad smells affect overall performance and quality of application. So, it’s necessary to reduce this smells counts as low as possible. Yeah, it’s not possible to remove completely practically. Here we have suggested some refactor methods that we can apply to bad code smells to remove it or reduce it effects.

* **Extract class:** Sometimes one class carrying too many responsibilities and create a complex structure called God Class. This complex structure makes maintenance activity very hard. Thanks to Extract Class refactoring method that can resolve this complexity issue to some extent. Extract class method helps establish SRP (Single Responsibility Principle). In this method we have to move methods and attributes to new class that are irrelevant to old class’s responsibility. To apply this method we have to follow some steps:
  + First, you have to decide how to divide responsibilities.
  + Create a new class and give a reference link to this class in old class.
  + Now first start with Data fields to move and then move methods.
  + Compile and test after each move.
  + If you need an older class link in new class then try to avoid it because it creates dependency issues.
* **Move Method:** This Refactoring method is used to move method or data blocks to some other class that is more relevant to its functioning. To use this method sometimes you need to use **Extract Method** Refactoring first to extract some part of a method or data block that you think might be more linked to or suitable to some other class. This method can be used to remove Feature Envy, Long Method etc. type of bad smells. Here are steps to follow to apply this method:
  + Analyze all the features in target class that are used by method in the source class and on that basis decide whether to move it to target class or not.
  + Check other declarations in the super or subclass. If it exists, then it’s not advisable to move method.
  + Declare method in target class and copy and paste code into it.
  + Make source method delegating method or delete it and handle references manually.
  + Compile and test every time you make changes.

**Code snippet in Source Code**

This snippet is from UserAgent.java class from useragent package. We found this part of code which is unrelated to class’s respondibility. Also, this class is god class containing so much different responsibility into it. We can move this code into separate new class to reduce its responsibility.





