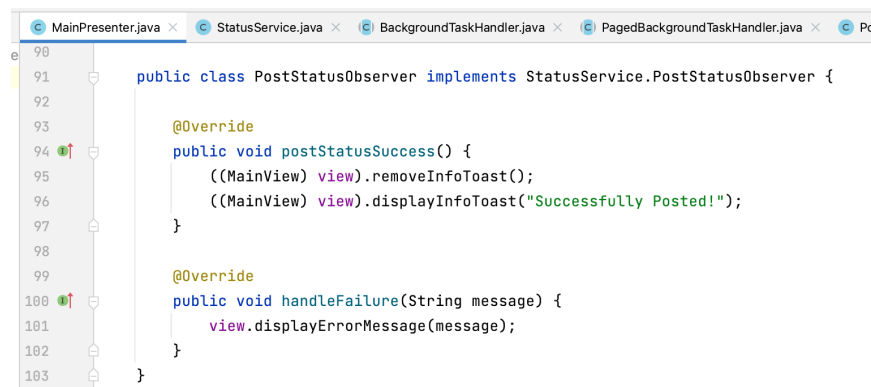


## Milestone Report

1. Pick one place where you used the observer pattern. Which class was the subject? Which class was the observer? Which layer did the subject belong to and which layer did the observer belong to? (Model, View, or Presenter layer)

```
public class StatusService extends ServiceTemplate {  
  
    public interface PostStatusObserver extends ServiceTemplate.ServiceObserver {  
        void postStatusSuccess();  
    }  
}
```



```
98  
99  
100 public class PostStatusObserver implements StatusService.PostStatusObserver {  
101  
102     @Override  
103     public void postStatusSuccess() {  
104         ((MainView) view).removeInfoToast();  
105         ((MainView) view).displayInfoToast("Successfully Posted!");  
106     }  
107  
108     @Override  
109     public void handleFailure(String message) {  
110         view.displayErrorMessage(message);  
111     }  
112 }  
113
```

The PostStatusObserver interface is located as part of the Model layer, in the StatusService class. However, it is implemented in the Presenter layer, in the MainPresenter class. The PostStatusObserver is the observer class (officially it is in the presenter layer), and it observes the PostStatusHandler (which is part of the model layer). In this way, the presenter layer is notified when the post status handler is done handling a message.

2. Pick one place where you used generics. What class was it in? What classes can the generic type T be?

```
public abstract class PagedBackgroundTaskHandler<T> extends ServiceTemplate.PagedServiceObserver<  
    extends BackgroundTaskHandler<T> {  
  
    public PagedBackgroundTaskHandler(ServiceTemplate.ServiceObserver observer) {  
        super((T) observer);  
    }  
  
    @Override  
    protected void handleSuccessMessage(ServiceTemplate.PagedServiceObserver observer, Bundle data) {  
        List<T> items = (List<T>) data.getSerializable(PagedTask.ITEMS_KEY);  
        boolean hasMorePages = data.getBoolean(PagedTask.MORE_PAGES_KEY);  
        observer.getItemsSuccess(items, hasMorePages);  
    }  
}
```

I used generic types in several different classes, especially those related to story/followers/following/feed, which are all paginated. In this case, the class is the PagedBackgroundTaskHandler, which inherits from the abstract BackgroundTaskHandler class. In the PagedBackgroundTaskHandler, the method overriding handleSuccessMessage() has a generic type T. This generic type T can either be User or Status.

3. Pick one place where you used the template method pattern. Show the template method. What class is it in? Show the step of the algorithm that is deferred to the class's subclass. What class is it in?

```
public void initiateAuth(String firstName, String lastName, String username,
                        String password, ImageView imageToUpload) {
    String message = validateInputs(firstName, lastName, username,
                                    password, imageToUpload);

    ((AuthView) view).clearErrorMessage();

    if (message == null) {
        ((AuthView) view).displayInfoMessage(getInfoMessage());

        authSuccess(firstName, lastName, username,
                    password, imageToUpload);
    } else {
        ((AuthView) view).clearInfoMessage();
        view.displayErrorMessage(message);
    }
}
```

! Git is not installed  
Cannot identify the repository  
[Download and install Git](#)

```
public abstract void authSuccess(String firstName, String lastName, String username,
                                String password, ImageView imageToUpload);
```

This initiateAuth() method is found in the AuthPresenter class. The method authSuccess() is templated in AuthPresenter, but then it is later implemented in subclasses, such as LoginPresenter.

```
public class LoginPresenter extends AuthPresenter {  
  
    public LoginPresenter(AuthView loginView) { super(loginView); }  
  
    @Override  
    public void authSuccess(String firstName, String lastName, String username,  
                            String password, ImageView imageToUpload) {  
        new UserService().login(username, password, loginObserver: this);  
    }  
}
```