1. Facebook login은 Parse.com에서 ParseFaceBookUtils 클래스를 지원하고 있습니다. 구현된 소스는 비공개이며, API에서는 사용법에 대한 설명만 나와 있습니다. 따라서 Facebook 에서 지원하는 SDK내부의 login과 관련된 소스를 분석하여 Authentication과 Authorization에 대해 조사해보려고 하였습니다.

[요약]

ⅰ) Facebook AccessToken

- expires : 만기일
- permissions : 권한
- declinedPermissions : declined 권한
- token : 실제 facebook 으로 얻은 token 값.
- source : AccessTokenSource.FACEBOOK_APPLICATION, AccessTokenSource.WEB_VIEW 등의 enum 값
- lastRefresh : 가장 최근의 refresh Time
- applicationid : AccessToken과 관련된 Facebook Application의 Id
- userId : 유저의 Id 입니다.

ⅱ) Facebook AccessToken Request

- loginBehavior : 실제 로그인할 때 로그인 방식을 설정, SSO_WITH_FALLBACK(기본), SSO_ONLY, SUPPRESS_SSO
- permissions : read, pulish 등의 권한 설정
- defaultAudience : Facebook의 공개범위. 감추기, 혼자만 보기, 친구에게 보이기(기본), 모두에게 보이기
- applicationId : com.facebook.sdk.ApplicationId의 metaData 중 문자열이나 숫자로 된 sdk 버전.
- authId : UUID(범용 공유 식별자)값, RFC 4122(http://www.ietf.org/rfc/rfc4122.txt)참고.

ⅲ) Facebook AccessToken Result

- code : SUCCESS, CANCEL, ERROR 의 enum 값
- token : 실제 반환받은 AccessToken
- errormessage : 에러메시지 ┐ 해당 부분은 정리하지 못하였습니다.
- errorCode : 에러코드 ┘
- request : result를 반환하게 한 request - 정확하지 못합니다.
- loggingExtras : - 정확하지 못합니다.

추가: 페이스북 모바일 AccessToken의 경우 만기일이 대개 60일 이라고 합니다.

Package	com.facebook.login
└Class	LoginFragment.java
└Class	LoginClient.java
└Class	LoginManager.java
└Class	LoginLogger.java
Package	com.facebook
└Class	FacebookActivity.java
Package	com.facebook.appevents
└Class	AppEventsLogger

public class FacebookActivity extends FragmentActivity

```
...(생략)...
private static String FRAGMENT_TAG = “SingleFragment”;
private Fragment singleFragment;
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  Intent intent = getIntent();
  ...(생략)...
  Fragmentmanager manager = getSupportFragmentManager();
  Fragment fragment = manager.findFragmentByTag(FRAGMENT_TAG);
```

```
  if( fragment == null ) {
    if( FacebookDialogFragment.TAG.equlas(intent.getAction()) ) { // TAG = “ FacebookDialogFragment “
      FacebookDialogFragment dialogFragment = new FacebookDialogFragment();
      dialogFragment.setRetainInstance(true);
      dialogFragment.show(manager, FRAGMENT_TAG);
      fragment = dialogFragment;
    } else { // TAG = “ LoginFragment ”
      fragment = new LoginFragment();
      fragment.setRetainInstance(true);
      manager.beginTransaction()
        .add(R.id.com_facebook_fragment_container, fragment, FRAGMENT_TAG)
        .commit();
    }
  }
  singleFragment = fragment;
}
```

LoginFragment 가 실행될 Activity 입니다.
Fragment의 TAG로 실행할 Fragment가 무엇인지 구분합니다.

```
public class LoginFragment extends Fragment
...(생략)...
static final String RESULT_KEY = “ com.facebook.LoginFragment:Result ”;
private static final String TAG = “ LoginFragment ”;
private static final String SAVED_LOGIN_CLIENT = “ loginClient ”;
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  if( savedInstanceState != null ) { // not first login
    loginClient = savedInstanceState.getParcelabel(SAVED_LOGIN_CLIENT);
    loginClient.setFragment(this);
  } else { // first login
    loginClient = new LoginClient(this);
  }
  loginClient.setOnCompletedListener(new LoginClient.OnCompletedListener() {
    @Override
    public void onCompleted(LoginClient.Result outcome) {
      onLoginClientCompleted(outcome);
    }
  });
  ...(생략)...
}
private void onLoginClientCompleted(LoginClient.Result outcome) {
  request = null;
  int resultCode = (outcome.code == LoginClient.Result.Code.CANCEL) ? // Result is inner class,
    Activity.RESULT_CANCELED : Activity.RESULT_OK;                    // Result.Code is enum
  Bundle bundle = new Bundle();
  bundle.putparcelable(RESULT_KEY, outcome);
  Intent resultIntent = new Intent();
  resultIntent.putExtra(bundle);
  // The activity might be detached we will send a cancel result in onDetach
  if( isAdded() ) {
    getActivity().setResult(resultCode, resultIntent);
    getActivity().finish();
  }
}
```

```
@Override
public void onSavedInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  outState.putParcelable(SAVED_LOGIN_CLIENT, loginClient); // not first login
}
```

TAG가 LoginFragment인 Fragment 입니다.
Fragment 생명주기 중 onCreate에 LoginClient를 생성합니다.
LoginClient는 Parcelable 객체이기 때문에 Bundle형식으로 저장되어 메모리가 여유 있는 한 읽을 수 있습니다.
Facebook 로그인 중 Callback 의 결과로 Result를 얻게 되는데, 이 때 상태코드를 판별하여 저장합니다.

```
class LoginClient implements Parcelable
...(생략)...
public interface OnCompletedListener {
  void onCompleted(Result result);
}
void setFragment(Fragment fragment) {
  if( this.fragment != null ) {
    throw new FacebookException( " Can ' t set fragment once it is already set. " );
  }
  this.fragment = fragment;
}
public static class Request implements Parcelable {
  private final LoginBehavior loginBehavior; // enum
  private Set<String> permissions;
  private final DefaultAudience defaultAudience; // enum
  private final String applicationId;
  private final String authId;
  private boolean isRerequest = false;
  ...(생략)..
}
public static class Result implements Parcelable {
  enum Code {
    SUCCES( " success " ), CANCEL( " cancel " ), ERROR( " error " );
    private final String loggingValue;
    Code(String loggingValue) { this.loggingValue = loggingValue; }
    String getLoggingValue() { return loggingValue; }
  }
  final Code code;
  final AccessToken token;
  final String errorMesage;
  final String errorCode;
  final Request request;
  public Map<String, String> loggingExtra;
  ...(생략)...
}
```

LoginFragment를 지정할 수 있으며 LoginCompleteListner 인터페이스를 정의할 수 있도록 했습니다.
Request에는 loginBehavior : Specifies the behaviors to try during login. // enum 입니다.
  └ SSO_WITH_FALLBACK(true, true), SSO_ONLY(true, false), SUPPRESS_SSO(false, true)
  defaultAudience : NONE(null), ONLY_ME, FRIENDS, EVERYONE // enum입니다.
  permissons, applicationId, authId, isRerequest 로 이루어져있습니다.
Result에는 Code : SUCCESS, CANCEL, ERROR // enum입니다.
  logginValue, errorMessage, errorCode, request, loggingExtra로 이루어져있습니다.
LoginClient는 LoginManager에서 관리되며 실제 로그인에 관련된 부분은 LoginLogger를 통해 이루어집니다.

```
public class LoginManager
```

```
...(생략)...
private static volatile LoginManager instance;
private LoginClient.Request pendingLoginRequest;
private HashMap<String, String> pendingLoggingExtras;
private Context context;
private LoginLogger loginLogger;
public static LoginManager getInstance() {
  if( instance == null ) {
    synchronizaed ( LoginManager.class ) {
      if( instance == null ) instance = new LoginManager();
    }
  }
  return instance;
}
boolean onAcivityResult(int resultCode, Intent data, FacebookCallback<LoginResult> callback) {
  if( pendingLoginRequest == null ) {
    return false;
  }
  FacebookException exception = null;
  AccessToken newToken = null;
  LoginClient.Result.Code code = LoginClient.Result.Code.ERROR;
  Map<String, String> loggingExtra = null;
  boolean isCanceled = false;
  if( data != null ) {
    LoginClient.Result result = (LoginClient.Result)
      data.getParcelableExtra(LoginFragment.RESULT_KEY); // RESULT_CANCELED or RESLT_OK
    if( result != null ) {
      code = result.code;
      if( resultCode == Activity.RESULT_OK ) {
        if( result.code == LoginClient.Result.Code.SUCCESS ) {
          newToken = result.token;
        } else {
          exception = new FacebookAuthorizationException(result.errorMesage);
        }
      } else if( resultCode == Activity.RESULT_CANCELED ) {
        isCanceled = true;
      }
      loggindExtras = result.loggingExtras;
    }
  } else if( resultCode == Activity.RESULT_CANCELED )  { // data == null
    isCanceled = true;
    code = LoinClient.Result.Code.CANCEL;
  }
  if( exception == null && newToken == null && !isCanceled ) {
    exception = new FacebookException( " Unexpected call to LoginManager.onActivityResult " );
  }
  logCompletedLogin(code, loggingExtras, exception);
  finishLogin(newToken, exception, iscanceled, callback);
  return true;
}
private void logCompleteLogin(LoginClient.Result.Code result, Map<String, String> resultExtras,
  Exception exception) {
  if( pendingLoginRequest == null ) {
    getLogger().logUnexpectedError()(
      LoginLogger.EVENT_NAME_LOGIN_COMPLETE,
```

```
      “Unexpectedd call to logCompleteLogin with null pendingAuthorizationRequest.”
    );
  } else {
    getLogger().logCompleteLogin(
      pendingLoginRequest.getAuthId(),
      pendingLoggingExtras,
      result,
      resultExtras,
      exception);
  }
}
private void finishLogin(AccessToken newToken, FacebookException exception, boolean isCanceled,
  FacebookCallback<LoginResult> callback) {
  if( newToken != null ) {
    AccessToken.setCurrentAccessToken(token);
    Profile.fetchProfileForCurrentAccessToken();
  }
  if( callback != null ) {
    LoginResult loginResult = new Token != null
      ? computeLoginResult(pendingLoginRequest, newToken)
      : null;
    if(isCanceled || (loginResult != null && loginResult.getRecentlyGrantedPermissions().size()==0)){
      callback.onCancel();
      return;
    }
    if( exception != null ){
      callback.onError(exception);
    } else if( newToken != null ) {
      callback.onSuccess(loginResult);
    }
  }
  this.context = null;
  this.loginLogger = null;
}
private LoginLogger getLogger() {
  if( loginLogger == null ||
       !loginLogger.getApplicationId().equals( pendingLoginRequest.getApplicationId())) {
    loginLogger = new LoginLogger( context, pendingLoginRequest.getApplicationId());
  }
  return loginLogger;
}
// StartActivityDelegate is inner class
private void startLogin( StartActivityDelegate startActivityDelegate, LoginClient.Requet request )
  throws FacebookException {
  this.pendingLoginRequest = request;
  this.pendingLoggingExtras = new HashMap<>();
  this.context = startActivityDelegate.getActivityContext();
  logStartLogin();
  CallbackmanagerImpl.registerStaticCallback(
    CallbackManagerImpl.RequestCodeOffset.Login.toRequestCode(),
    new CallbackmanagerImplCallback() {
      @Override
      public boolean onActivityResult(int resultCode, Intent data) {
        return LoginManager.this.onActivityResult(resultCode, data);
      }
```

```
    }
  );
  boolean started = tryFacebookActivity(startActivityDelegate, request);
  pendingLoggingExtras.put(
    LoginLogger.EVENT_EXTRAS_TRY_LOGIN_ACTIVITY,
    started ?
    AppEventsConstanst.EVENT_PARAM_VALUE_YES : AppEventsConstants.EVENT_PARAM_VALUE_NO
  );
  if ( !started ) {
    FacebookException exception = new FacebookException(
      "Log in attempt failed: FacebookActivity could not be started." +
        "Please make sure you added FacebookActivity to the AndroidManifest.");
    logCompleteLogin(LoginClient.Request.Code.ERROR, null, exception);
    this.pendingLoginRequest = null;
    throw exception;
  }
}
private void logStartLogin() {
  getLogger().logStartLogin(pendingLoginRequest);
}
private boolean tryFacebookActivity(StartActivityDelegate startActivityDelegate,
  LoginClient.Request request) {
  Intent intent = getFacebookActivityIntent(request);
  if( !resolveIntent(intent)) {
    return false;
  }
  try {
    startActivityDelegate.startActivityForResult(intent, LoginClient.getLoginReqeustCode());
  } catch ( ActivityNotFoundException e) {
    return false;
  }
  return true;
}
private boolean resolveIntent(Intent intent) {
  ResolveInfo resolveInfo = Facebook.getApplicationContext().getPackageManager()
    .resolveActivity(intent, 0);
  if( resolveInfo == null ){
    return false;
  }
  return true;
}
public void logInWithReadPermissions(Fragment fragment, Collection<String> permissions) {
  validateReadPermissions(permissions);
  LoginClient.Request loginReqeust = createLoginRequest(permissions);
  startLogin(new FragmentStartActivityDelegate(fragment), loginRequest);
}
public void logInWithReadPermissions(Activity activity, Collection<String> permissions) {
  validateReadPermissions(permissions);
  LoginClient.Request loginRequest = createLoginReqeust(permissions);
  startLogin(new ActivityStartActivityDelegate(activity), loginRequest);
}
public void validateReadPermissions(Collection<String> permissions) {
  if( permissions == null ) {
    return;
  }
```

```
  for( String permission : permissions ){
    if( isPublishPermission(permission) ) {
      throw new FacebookException(
        String.format( " Cannot pass a publish or manage permission (%s) to a request for read "
          + " authorization " , permission));
    }
  }
}
private LoginClient.Request createLoginRequest(Collection<String> permissions) {
  LoginClient.Request request = new LoginClient.Request(
    loginBehavior,
    Collections.unmodifiableSet( permissions != null ? new HashSet(permissions)
      : new HashSet<String>()),
    defaultAudience,
    FacebookSdk.getApplicationId(),
    UUID.randomUUID().toString()
  );
  request.setRerequest(AccessToken.getCurrentAccessToken != null );
  return request;
}
```

LoginManager는 volatile 이고 동시성을 해결하기 위해 getInstance를 보시면 class자체를 synchronized 합니다.
onAcivityResult에서는 facebook callback login 과정을 마친 뒤 앞서 저장한 상태코드를 판별하여 로그인을 마무리 하는 과정입니다.
logCompleteLogin에서는 로그인 상태코드에 이상이 없을 경우 LoginLogger를 통해 Login을 진행하게 됩니다.
finishLogin에서는 AccessToken과 Profile을 저장하게 됩니다.

```
  AccessToken은 Date expires
                Set<String> permissions
                Set<String> declinedPermissions
      (NotNull) String token, the access token string obtained from Facebook
                AccessTokenSource source, an enum indicating how the token was originally obtained(
                                          in most cases, this will be either AccessTokenSource.FACEB
                                          OOK_APPLICATION or AccessTokenSource.WEB_VIEW); if null,
                                          FACEBOOK_APPLICATION is assumed.
                Date lastRefresh
     (NotNull)  String applicationId, the ID of the Facebook Application associated with this
                                       accesstoken
     (NotNull)  String userId, the id of user
  Profile은 String id, (NotNull) The id of the profile.
            String firstName
            String middleName
            String lastName
            String name
            Uri linkUri, The link for this profile. Can be null.
```

으로 이루어져있습니다.
이 외에 함수들은 ReadPermission을 가지고 로그인하는 것들과 관련된 함수들만 모아보았습니다.

LoginLogger

```
private final AppEventsLogger appEventsLogger;
...(생략)...
static final String EVENT_PARAM_AUTH_LOGGER_ID = " 0_auth_logger_id ";
static final String EVENT_PARAM_TIMESTAMP = " 1_timestamp_ms ";
static final String EVENT_PARAM_LOGIN_RESULT = " 2_result ";
static final String EVENT_PARAM_METHOD= " 3_method ";
...(생략)...
```

```
static Bundle newAuthorizationLoggingBundle(String authLoggerId) {
  Bundle bundle = new Bundle();
  bundle.putLong(EVENT_PARAM_TIMESTAMP, System.currentTimeMillis());
  bundle.putString(EVENT_PARAM_AUTH_LOGGER_ID, authLoggerId);
  bundle.putString(EVENT_PARAM_METHOD, “ ”);
  bundle.putString(EVENT_PARAM_LOGIN_RESULT, “ ”);
  bundle.putString(EVENT_PARAM_ERROR_MESSAGE, “ ”);
  bundle.putString(EVENT_PARAM_ERROR_CODE, “ ”);
  bundle.putString(EVENT_PARAM_EXTRAS, “ ”);
  return bundle;
}
public void logStartLogin(LoginClient.Request pendingLoginReqeust ) {
  Bundle bundle = newAuthorizationLoggingBundle(pendingLoginRequest.getAuthId());
  try {
    JSONObject extras = new JSONObject();
    extras.put(EVENT_EXTRAS_LOGIN_BEHAVIOR,
      pendingLoginRequest.getLoginBehavior().toString());
    ...(생략)...
  }
  appEventsLogger.logSdkEvent(EVENT_NAME_LOGIN_STATRT, null, bundle);
}
```

login에 관련된 constants들이 차례로 정의되어 있으며 이를 활용하여 bundle을 만듭니다.
만든 bundle을 가지고 해당하는 log 이름과 함께 logSdkEvent를 실행합니다.

AppEventsLogger

```
public void logSdkEvent(String eventname, Double valueToSum, Bundle parameters) {
  logEvent(eventName, valueToSum, parameters, true);
}
private void logEvent(String eventName, Double valueToSum, Bundle parameters,
  boolean isImplicitlyLogged ) {
  AppEvent event = new AppEvent(this.context, eventName, valueToSum, parameters, isImplicitlyLogged);
  logEvent(context, event, accessTokenAppId);
}
private static void logEvent(final Context context, final AppEvent event,
  final AccessTokenAppIdPair accessTokenAppId) {
  FacebookSdk.getExecutor().execute(new Runnable() {
    @Override
    public void run() {
      SessionEventsState state = getSessionEventsState(context, accessTokenAppId);
      state.addEvent(event);
      flushIfNecessary();
    }
  });
}
```

LoginLogger의 bundle을 통해 넘겨받은 파라미터들(constants)을 통해 실질적으로 로그인을 실행합니다.