The source code for the EvaluationInjector.java, with an example of how to initialize it

```
import androidx.fragment.app.FragmentActivity;
import androidx.fragment.app.Fragment;
import android.app.Activity;
import android.view.ViewParent;
import android.view.ViewTreeObserver;
import android.app.Application;
import android.graphics.Rect;
import android.os.Bundle;
import android.util.Log;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.recyclerview.widget.RecyclerView;
import androidx.fragment.app.FragmentManager;
import androidx.drawerlayout.widget.DrawerLayout;
import com.google.android.material.navigation.NavigationView;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.text.SimpleDateFormat;
import java.util.*;
import android.util.DisplayMetrics;
import android.graphics.Bitmap;
import android.graphics.Canvas;

public class EvaluationInjector {
   private static String sessionId;
   private static File logFile;
   // directory for all screenshots
   private static File screenshotDir;
   // remember which screens we've already captured
   private static final Set<String> seenScreens = new HashSet<>();
   // counter for naming screenshot files
   private static int globalScreenshotCounter = 1;
   private static final Set<FragmentActivity> fragmentListeners = new HashSet<>();
```

```
private static int swipeCounter = 1;
private static int tapCounter = 1;

private static final Map<String, TrackedItem> swipedItems = new HashMap<>();
private static Activity currentActivity;
private static final int SWIPE_THRESHOLD = 100;
private static String lastTapItemId = "";
// used to suppress double-tap noise
private static long lastTapTimestamp = 0;
// to track a potential tap vs. a swipe
private static float  touchStartX;
private static float  touchStartY;
private static boolean isSwipeGesture;
private static boolean skipNextScreenshot = false;
/**
 * Core logging routine for any tap-like event.
 */
private static void logTapEvent(
      String rawItemId,
      String uiElementType,
      String labelContext,
      String pkg,
      String visibleUi,
      int touchX,
      int touchY,
      Rect bbox,
      String zone
) {
   // 1) normalize + dedupe
   long now = System.currentTimeMillis();
   if (rawItemId.equals(lastTapItemId) && now - lastTapTimestamp < 300) {
      return;  // too-quick duplicate
   }
   lastTapItemId    = rawItemId;
   lastTapTimestamp = now;

   // 2) assign tap ID
   String tapId = "tap_" + (tapCounter++);

   // 3) build + write
   String entry = formatTimestamp() +
         " | TAP_EVENT" +
         " | tap_id: "      + tapId +
         " | item_id: "     + rawItemId +
         " | ui_element: "   + uiElementType +
         " | label_context: "+ labelContext +
         " | package: "     + pkg +
```

```
        // new fields:
        " | coords: ("     + touchX + "," + touchY + ")" +
        " | bbox: ("       + bbox.left + "," + bbox.top +
        "," + bbox.right + "," + bbox.bottom + ")" +
        " | zone: "       + zone +
        " | visible_ui: "   + visibleUi;


    writeLog(entry);
}
    /** Call once from Application.onCreate() */
public static void init(Application app) {
    sessionId = UUID.randomUUID().toString();
    logFile = new File(app.getFilesDir(), "logfile.txt");
    try (FileOutputStream fos = new FileOutputStream(logFile, false)) {
        fos.write(("New Session Started (" + sessionId + ") ===\n").getBytes());
    } catch (IOException e) {
        Log.e("Logger", "Failed to clear log", e);
    }
    // — setup screenshots folder —
    screenshotDir = new File(app.getFilesDir(), "screenshots");
    if (screenshotDir.exists()) {
        for (File f : screenshotDir.listFiles()) {
            f.delete();
        }
    } else {
        screenshotDir.mkdirs();
    }
    // reset screenshot state
    seenScreens.clear();
    globalScreenshotCounter = 1;


    app.registerActivityLifecycleCallbacks(new Application.ActivityLifecycleCallbacks() {
        @Override public void onActivityCreated(Activity a, Bundle s) {}
        @Override public void onActivityStarted(Activity a) { hook(a); }
        @Override public void onActivityResumed(Activity a) { hook(a);
        }
        @Override public void onActivityPaused(Activity a) {}
        @Override public void onActivityStopped(Activity a) {}
        @Override public void onActivitySaveInstanceState(Activity a, Bundle o) {}
        @Override public void onActivityDestroyed(Activity a) {}
    });
}

/** Replaces the Activity's Window.Callback to intercept taps, and installs swipe detectors. */
private static void hook(Activity activity) {
    currentActivity = activity;
```

```
String pkg = activity.getPackageName();

// 1) wrap dispatchTouchEvent via dynamic proxy
Window win = activity.getWindow();
Window.Callback orig = win.getCallback();
Window.Callback proxy = (Window.Callback) Proxy.newProxyInstance(
    Window.Callback.class.getClassLoader(),
    new Class[]{ Window.Callback.class },
    new InvocationHandler() {
      @Override
      public Object invoke(Object proxyObj, Method method, Object[] args)
          throws Throwable {
        String name = method.getName();

        // 1) TAP_EVENT & swipe-vs-tap logic (exactly as before)
        if ("dispatchTouchEvent".equals(name)
            && args != null
            && args.length == 1
            && args[0] instanceof MotionEvent) {
          MotionEvent ev = (MotionEvent) args[0];
          switch (ev.getActionMasked()) {
            case MotionEvent.ACTION_DOWN:
              touchStartX  = ev.getX();
              touchStartY  = ev.getY();
              isSwipeGesture = false;
              break;
            case MotionEvent.ACTION_MOVE:
              if (Math.abs(ev.getX() - touchStartX) > SWIPE_THRESHOLD
                  || Math.abs(ev.getY() - touchStartY) > SWIPE_THRESHOLD) {
                isSwipeGesture = true;
              }
              break;
            case MotionEvent.ACTION_UP:
              if (!isSwipeGesture) {
                View decor  = activity.getWindow().getDecorView();
                View tapped = findViewAt(decor,
                    (int)ev.getX(),
                    (int)ev.getY());
                handleTap(tapped, ev, pkg, activity);
              }
              break;
          }
        }

        // 2) Menu-item taps (unchanged)
        if (args != null) {
          for (Object a : args) {
            if (a instanceof android.view.MenuItem) {
```

```
                handleMenuItem((android.view.MenuItem)a, pkg, activity);
                break;
              }
            }
          }

          // 3) Call through to the real Window.Callback
          Object result = method.invoke(orig, args);

          if ("onWindowFocusChanged".equals(method.getName())
              && args != null
              && args.length == 1
              && args[0] instanceof Boolean) {
            boolean hasFocus = (Boolean) args[0];

            if (!hasFocus) {
              // our window lost focus (e.g. permission popup)
              skipNextScreenshot = true;
            } else if (skipNextScreenshot) {
              // it just returned—take one delayed shot
              View decor = activity.getWindow().getDecorView();
              decor.postDelayed(() -> takeScreenshot(activity), 300);
              skipNextScreenshot = false;
            }
          }
          // 5) Return the original result (only one return!)
          return result;
        }
      }

);

win.setCallback(proxy);

// 2) install swipe detectors recursively on content view children
ViewGroup content = activity.findViewById(android.R.id.content);
attachSwipeListeners(content, pkg, activity);

if (activity instanceof FragmentActivity) {
  FragmentActivity fa = (FragmentActivity) activity;
  // register only once per Activity instance
  if (fragmentListeners.add(fa)) {
    fa.getSupportFragmentManager().registerFragmentLifecycleCallbacks(
        new FragmentManager.FragmentLifecycleCallbacks() {


          @Override
          public void onFragmentResumed(FragmentManager fm, Fragment f) {
```

```
                View fragRoot = f.getView();
                Runnable take = () -> {
                  // only fire if the window is focused _and_ we're not in "skipNextScreenshot"
mode
                  if (activity.hasWindowFocus() && !skipNextScreenshot) {
                    takeScreenshot(activity);
                  }
                };
                if (fragRoot != null) {
                  fragRoot.postDelayed(take, 500);
                } else {
                  activity.getWindow().getDecorView()
                      .postDelayed(take, 500);
                }
              }

            },
            true  // listen also to child-fragment events
        );
      }
    }

  }

  /** Recursively attach swipe detectors to every child view. */
  private static void attachSwipeListeners(ViewGroup root, String pkg, Activity act) {
    for (int i = 0; i < root.getChildCount(); i++) {
      View child = root.getChildAt(i);
      GestureDetector sd = createSwipeDetector(child, pkg, act);
      child.setOnTouchListener((v, ev) -> {
        sd.onTouchEvent(ev);
        return false;
      });
      if (child instanceof ViewGroup) {
        attachSwipeListeners((ViewGroup) child, pkg, act);
      }
    }
  }

  private static GestureDetector createSwipeDetector(View target, String pkg, Activity act) {
    return new GestureDetector(
        target.getContext(),
        new GestureDetector.SimpleOnGestureListener() {
          // holds the exact container you started swiping on
          private View boundsOwner;

          @Override
          public boolean onDown(MotionEvent e) {
```

```
    // 1) find the view under your finger
    View decor = act.getWindow().getDecorView();
    View tapped = findViewAt(decor,
            (int)e.getRawX(),
            (int)e.getRawY());

    // 2) climb to its card/container (or stay on the view)
    ViewGroup itemContainer = findItemContainer(tapped);
    boundsOwner = (itemContainer != null) ? itemContainer : tapped;

    return true;  // must return true to receive onFling
}

@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float vx, float vy) {
    float dx = e2.getX() - e1.getX(), dy = e2.getY() - e1.getY();
    String dir = null;
    if (Math.abs(dx) > Math.abs(dy) && Math.abs(dx) > SWIPE_THRESHOLD) {
        dir = dx > 0 ? "right" : "left";
    } else if (Math.abs(dy) > SWIPE_THRESHOLD) {
        dir = dy > 0 ? "down" : "up";
    }
    if (dir == null) return false;
    final String swipeDir = dir;

    View itemView = boundsOwner != null ? boundsOwner : target;
    List<String> parts = new ArrayList<>();
    collectTextsForSwipe(itemView, parts);
    parts.removeIf(String::isEmpty);
    if (parts.isEmpty()) return false;
    if (parts.size() > 3) parts = parts.subList(0, 3);
    String label = String.join(" | ", parts);

    String itemId  = generateItemId(label, itemView);
    String ui      = getVisibleScreenName(act);
    String swipeId = "swipe_" + (swipeCounter++);

    // immediate log
    TrackedItem ti = new TrackedItem(swipeId, label, itemId, ui, pkg);
    swipedItems.put(itemId, ti);
    Rect bbox = new Rect();
    itemView.getGlobalVisibleRect(bbox);
    int touchX = (int) e2.getRawX();
    int touchY = (int) e2.getRawY();

    DisplayMetrics dm = act.getResources().getDisplayMetrics();
    int sw = dm.widthPixels, sh = dm.heightPixels;
    String vert = (touchY < sh/3)    ? "TOP"
```

```
                    : (touchY > sh*2/3) ? "BOTTOM"
                    : "MIDDLE";
            String horz = (touchX < sw/3)    ? "LEFT"
                    : (touchX > sw*2/3) ? "RIGHT"
                    : "CENTER";
            String zone = vert + "_" + horz;

            writeLog(buildSwipeLog(
                    ti, swipeDir,
                    touchX, touchY, bbox, zone));


            return true;
          }

        }
    );
}

/** Recursively find the deepest child under (x,y) within this root. */
private static View findViewAt(View root, int x, int y) {
    if (!(root instanceof ViewGroup)) return root;
    ViewGroup g = (ViewGroup) root;
    for (int i = g.getChildCount() - 1; i >= 0; i--) {
        View c = g.getChildAt(i);
        int[] loc = new int[2];
        c.getLocationOnScreen(loc);
        Rect r = new Rect(loc[0], loc[1], loc[0] + c.getWidth(), loc[1] + c.getHeight());
        if (r.contains(x, y)) {
            return findViewAt(c, x, y);
        }
    }
    return root;
}

private static void handleTap(View rawView, MotionEvent ev, String pkg, Activity act) {

    // 2) get the tapped view's own label (for label_context)
    String label = extractDeepLabel(rawView);
    if (label == null || label.trim().isEmpty()) {
        label = findLabelFromParents(rawView);
    }
    if (label == null || label.trim().isEmpty()) {
        label = "Unknown";
    }

    // 3) what screen are we on?
    String visibleUi = getVisibleScreenName(act);
```

```
// 4) find the enclosing container (card/row), if any
ViewGroup container = findItemContainer(rawView);

// 5) decide what text to use for item_id
String idKey;
if (container != null
      && (rawView instanceof Button
      || rawView instanceof ImageButton
      || rawView instanceof ImageView)) {
   // a) grab all text parts from the container
   List<String> parts = new ArrayList<>();
   collectDeepLabel(container, parts);   // your existing helper

   // b) remove the tapped view's own label via iterator
   Iterator<String> it = parts.iterator();
   while (it.hasNext()) {
      if (it.next().equals(label)) {
         it.remove();
      }
   }

   // c) if there's still something left, take the first piece
   if (!parts.isEmpty()) {
      idKey = parts.get(0);
   } else {
      // fallback to the container's full label
      idKey = extractDeepLabel(container);
   }
   // generate ID off the container's bounds
   rawView = container;
}
else if (container != null) {
   // taps elsewhere in a container: old behavior
   idKey = extractDeepLabel(container);
   rawView    = container;
}
else {
   // completely outside any container
   idKey = label;
}

if (idKey == null || idKey.trim().isEmpty()) {
   idKey = "Unknown";
}

// 6) build the item_id and ui_element
String itemId = generateItemId(idKey, rawView);
```

```
        String uiType = getUiElementType(rawView);
// — compute bounding box & touch coords —
        Rect bbox = new Rect();
        rawView.getGlobalVisibleRect(bbox);
        int touchX = (int) ev.getRawX();
        int touchY = (int) ev.getRawY();

// — compute zone tagging —
        DisplayMetrics dm = act.getResources().getDisplayMetrics();
        int sw = dm.widthPixels, sh = dm.heightPixels;
        String vert = (touchY < sh/3)    ? "TOP"
            : (touchY > sh*2/3) ? "BOTTOM"
            : "MIDDLE";
        String horz = (touchX < sw/3)    ? "LEFT"
            : (touchX > sw*2/3) ? "RIGHT"
            : "CENTER";
        String zone = vert + "_" + horz;

        // 7) log exactly as before
        logTapEvent(
            itemId, uiType, label, pkg, visibleUi,
            touchX, touchY, bbox, zone);
    }


    private static void handleMenuItem(android.view.MenuItem item, String pkg, Activity act) {
        String title = item.getTitle() != null
            ? item.getTitle().toString().trim()
            : String.valueOf(item.getItemId());
        String itemId = "menuitem_" + item.getItemId();
        String visibleUi = getVisibleScreenName(act);
        // for MenuItem taps we don't have real coords or bbox, so use placeholders
        int touchX = -1;
        int touchY = -1;
        Rect bbox   = new Rect(-1, -1, -1, -1);
        String zone  = "UNKNOWN";


        logTapEvent(
            itemId,       // rawItemId
            "MenuItem",    // uiElementType
            title,        // labelContext
            pkg,          // package
            visibleUi,     // visibleUi
            touchX, touchY,// coords
            bbox,         // bounding box
            zone
        );
```

```
  }


  private static String getUiElementType(View v) {
    if (v instanceof Button)       return "Button";
    if (v instanceof ImageButton)   return "ImageButton";
    if (v instanceof EditText)      return "TextField";
    if (v instanceof TextView)      return "Text";
    if (v instanceof ImageView)     return "Image";
    if (v instanceof DrawerLayout)  return "Drawer";
    if (v instanceof NavigationView) return "NavigationMenu";
    return v.getClass().getSimpleName();
  }

  /*———— Helpers for swipe-label extraction, deep-label extraction, container finding, etc. —
————*/

  private static String extractSwipeLabel(View view) {
    ViewGroup cg = findReasonableContainer(view);
    if (cg == null) {
      if (view instanceof TextView)
        return ((TextView) view).getText().toString().trim();
      return null;
    }
    List<String> parts = new ArrayList<>();
    collectTextsForSwipe(cg, parts);
    parts.removeIf(String::isEmpty);
    return parts.isEmpty() ? null : String.join(" | ", parts);
  }

  private static ViewGroup findReasonableContainer(View view) {
    ViewParent p = view.getParent();
    int depth = 0;
    while (p instanceof View && depth++ < 3) {
      if (p instanceof ViewGroup) return (ViewGroup) p;
      p = p.getParent();
    }
    return null;
  }

  private static void collectTextsForSwipe(View view, List<String> parts) {
    // stop at 3 pieces
    if (parts.size() >= 3) return;

    if (view instanceof TextView && !isActionWord(((TextView) view).getText().toString())) {
      String txt = ((TextView) view).getText().toString().trim();
      if (!txt.isEmpty()) parts.add(txt);
```

```
    } else if (view instanceof EditText) {
        String hint = ((EditText) view).getHint() != null
                ? ((EditText) view).getHint().toString().trim() : "";
        if (!hint.isEmpty()) parts.add(hint);

    } else if (view instanceof ImageView && view.getContentDescription() != null) {
        String d = view.getContentDescription().toString().trim();
        if (!d.isEmpty() && !isActionWord(d)) parts.add(d);
    }

    if (view instanceof ViewGroup) {
        ViewGroup g = (ViewGroup) view;
        for (int i = 0; i < g.getChildCount(); i++) {
            collectTextsForSwipe(g.getChildAt(i), parts);
            if (parts.size() >= 3) break;
        }
    }
}

private static boolean isActionWord(String t) {
    String l = t.toLowerCase(Locale.ROOT);
    return l.equals("restore") || l.equals("undo") || l.equals("add now")
            || l.equals("delete")  || l.equals("edit") || l.equals("ok")
            || l.equals("cancel")  || l.equals("yes")  || l.equals("no")
            || l.equals("retry")   || l.equals("done") || l.equals("submit")
            || l.equals("save")    || l.equals("send");
}

private static String extractDeepLabel(View view) {
    List<String> parts = new ArrayList<>();
    collectDeepLabel(view, parts);
    parts.removeIf(String::isEmpty);
    return parts.isEmpty() ? null : String.join(" | ", parts);
}

private static void collectDeepLabel(View view, List<String> parts) {
    if (view instanceof TextView) {
        String txt = ((TextView) view).getText().toString().trim();
        if (!txt.isEmpty()) parts.add(txt);
    } else if (view instanceof EditText) {
        String hint = ((EditText) view).getHint() != null
                ? ((EditText) view).getHint().toString().trim()
                : "";
        if (!hint.isEmpty()) parts.add(hint);
    } else if (view.getContentDescription() != null) {
        String d = view.getContentDescription().toString().trim();
        if (!d.isEmpty()) parts.add(d);
    }
```

```
    if (view instanceof ViewGroup) {
        ViewGroup g = (ViewGroup) view;
        for (int i = 0; i < g.getChildCount(); i++) {
            collectDeepLabel(g.getChildAt(i), parts);
        }
    }
}

private static ViewGroup findItemContainer(View view) {
    View current = view;
    for (int depth = 0; depth < 6; depth++) {
        if (!(current instanceof ViewGroup)) break;
        ViewGroup vg = (ViewGroup) current;

        if (isItemContainer(vg)) {
            return vg;
        }

        ViewParent p = vg.getParent();
        if (p instanceof ViewGroup) {
            current = (View) p;
        } else {
            break;
        }
    }
    return null;
}

private static boolean isItemContainer(ViewGroup vg) {
    int count = vg.getChildCount();
    if (count < 2 || count > 6) return false;

    // if every direct child is itself a ViewGroup, it's probably the list container
    boolean allGroups = true;
    for (int i = 0; i < count; i++) {
        if (!(vg.getChildAt(i) instanceof ViewGroup)) {
            allGroups = false;
            break;
        }
    }
    if (allGroups) return false;

    // ensure it has at least two real text bits (time + name, for example)
    List<String> parts = new ArrayList<>();
    collectTextsForSwipe(vg, parts);
    parts.removeIf(String::isEmpty);
    return parts.size() >= 2;
}
```

```
private static boolean hasMultipleContentChildren(ViewGroup group) {
  List<String> parts = new ArrayList<>();
  collectTextsForSwipe(group, parts);
  return parts.size() >= 2;
}

private static String findLabelFromParents(View view) {
  ViewParent p = view.getParent();
  int lvl = 0;
  while (p instanceof View && lvl++ < 3) {
    if (p instanceof ViewGroup) {
      List<String> lbls = new ArrayList<>();
      collectDeepLabel((View) p, lbls);
      if (!lbls.isEmpty()) return String.join(" | ", lbls);
    }
    p = p.getParent();
  }
  return null;
}

private static Set<String> getVisibleItemIds(View root) {
  Set<String> ids = new HashSet<>();
  if (root instanceof TextView) {
    String lbl = extractSwipeLabel(root);
    if (lbl != null) ids.add(generateItemId(lbl, root));
  } else if (root instanceof ViewGroup) {
    ViewGroup g = (ViewGroup) root;
    for (int i = 0; i < g.getChildCount(); i++) {
      ids.addAll(getVisibleItemIds(g.getChildAt(i)));
    }
  }
  return ids;
}

private static String generateItemId(String label, View view) {
  Rect b = new Rect();
  view.getGlobalVisibleRect(b);
  String norm = label.replaceAll("\\s+", "_").toLowerCase();
  return norm + "_" + b.left + "_" + b.top;
}

private static String buildSwipeLog(
    TrackedItem it,
    String dir,
    int touchX,
    int touchY,
    Rect bbox,
```

```
        String zone
    ) {
        return formatTimestamp() +
                " | SWIPE_EVENT" +
                " | swipe_id: "      + it.swipeId +
                " | package: "       + it.packageName +
                " | direction: "     + dir +
                " | swiped_ui_item: " + it.label +
                " | coords: ("       + touchX + "," + touchY + ")" +
                " | bbox: ("         + bbox.left + "," + bbox.top +
                "," + bbox.right + "," + bbox.bottom + ")" +
                " | zone: "          + zone +
                " | visible_ui: "    + it.visibleUi;
    }


    private static void writeLog(String txt) {
        try (FileOutputStream fos = new FileOutputStream(logFile, true)) {
            fos.write((txt + "\n").getBytes());
        } catch (IOException e) {
            Log.e("Logger", "Error writing log", e);
        }
    }

    private static String formatTimestamp() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS",
                Locale.getDefault());
        return sdf.format(new Date());
    }

    // — SCREENSHOT HELPERS —
    private static void takeScreenshot(Activity activity) {
        try {
            String visibleUi = getVisibleScreenName(activity);
           // if (!seenScreens.add(visibleUi)) return;

            int count = globalScreenshotCounter++;
            View root = activity.getWindow().getDecorView().getRootView();
            if (root.getWidth() == 0 || root.getHeight() == 0) return;

            Bitmap bmp = Bitmap.createBitmap(root.getWidth(), root.getHeight(),
                    Bitmap.Config.ARGB_8888);
            Canvas canvas = new Canvas(bmp);
            root.draw(canvas);

            File out = new File(screenshotDir,
                    visibleUi + "-SC" + count + ".png");
            try (FileOutputStream fos = new FileOutputStream(out)) {
```

```
            bmp.compress(Bitmap.CompressFormat.PNG, 100, fos);
        }
        Log.d("Screenshot", "Saved " + out.getName());
      } catch (Exception e) {
        Log.e("Screenshot", "failure", e);
      }
    }

    private static String getVisibleScreenName(Activity activity) {
      if (activity instanceof FragmentActivity) {
        FragmentActivity fa = (FragmentActivity) activity;
        for (Fragment f : fa.getSupportFragmentManager().getFragments()) {
          if (f != null && f.isVisible()) {
            // if this *is* a NavHostFragment unwrap it dynamically
            String className = f.getClass().getName();
            if ("androidx.navigation.fragment.NavHostFragment".equals(className)) {
              try {
                // getPrimaryNavigationFragment()
                Fragment child =
                    f.getChildFragmentManager()
                        .getPrimaryNavigationFragment();
                if (child != null && child.isVisible()) {
                  return child.getClass().getSimpleName();
                }
              } catch (Exception ignored) {}
            }
            // otherwise just return the fragment's own name
            return f.getClass().getSimpleName();
          }
        }
      }
      return activity.getClass().getSimpleName();
    }
    static class TrackedItem {
      String swipeId, label, itemId, visibleUi, packageName;
      TrackedItem(String id, String lbl, String item, String vu, String pkg) {
        swipeId = id;
        label = lbl;
        itemId = item;
        visibleUi = vu;
        packageName = pkg;
      }
    }
}
```

An example of how The EvaluationInjector is initialized within the class that extends the Android Application class:

```
package com.example.to_do_app;

import android.app.Application;

public class MyApp extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        EvaluationInjector.init(this);

    }
}
```