情報メディア特別演習 最終レポート

学籍番号	201211436
氏名	鳥羽雄希
アドバイザ教員氏名	志築文太郎
履修科目	情報メディア特別演習

演習テーマ名

おふとんリーディング：スキャンした本を楽に読むためのアプリケーション

第1章　はじめに

1.1　本演習の概要

本演習はJavaとAndroid開発の習得を目標としている。これまで自分は、C言語を学習するためにプログラムの断片を書いたことはあっても、実用的なアプリケーションを、何らかのプラットフォーム向けに開発した経験はなかった。複雑なAndroid SDKを使いこなすためにはどのように学習すべきかをこの演習を通して学び、iOSやMac、Windows、Linuxなど、他のプラットフォーム向けのアプリケーション開発につなげる。さらに、学習目的とはいえ約1年間かけて開発を行うのだから、自分が一番欲しいと思っていた、寝ながら本を読むためのアプリケーションを開発することとした。

1.2　おふとんリーディングの有用性

おふとんリーディングは、現在数多く公開されている読書に関するアプリケーション（以下「読書アプリ」と総称する）において無視されてきた観点から開発された。読書をする必要に迫られてはいるが、本を読むのは苦手な人のためのアプリケーションなのである。

これまでの読書アプリは、「本を読むこと自体は当然できる」と想定して開発されてきた。だから読むこと自体を補助するという考え方は薄かった。読むことについては文章を画面にそのまま表示するだけで、その他の高度な機能を実装することに力が注がれてきたと感じられる（例えば、翻訳する・コメントをソーシャルシェアリングする・複数端末間で読書記録を同期する・読書量をオンラインで競う…など)。たしかに、視覚障害者向けに、文字の大きさや画面のコントラストを調整したり、文章を読み上げたりする機能は、読書アプリに限らず以前からOSやWebブラウザにも搭載されてきた。しかし考えてみてほしい。視覚に問題さえなければ我々は誰でも当たり前に読書ができていたのだろうか。読書好きな人には信じられないかもしれないが、小説さえ字を追うことが面倒で読めない人はたくさんいる。ましてや教科書を集中して読める人がどれだけいるだろうか。しかも電子機器上では集中力を削ぐ要因が増える（他のアプリケーションを使いたい誘惑、バッテリーがもったいないと思う気持ち、透過光ディスプレイのまぶしさ、低解像度ディスプレイのモアレ…など）。

おふとんリーディングは、読むことを支援する方法も新しい。電光掲示板のように文字を流すのである。普段文章を読むことが苦手な人でも、電車などの電光掲示板やニコニコ動画のコメントなど、一度に目に入る文字数が非常に少なく、自分で丹念に視線を動かす必要がなければ、ついつい見入って全部読んでしまったという経験はないだろうか。

しかも、文章を読み上げる方法に比べて、電光掲示板のように文字を流す方法は汎用性が高い。文字行を画像として切り出し、それを流すように実装しているため、文字を認識する必要がないのである。読み上げる方法は文字情報が含まれていないと難しい。つまり電子書籍として売られている本にしか対応できない。ところが日本では紙でしか手に入らない本はまだまだ多い。紙の本を自分でスキャン（以下「自炊」と呼ぶ）した画像データから文字を読み上げるのは現実的ではない。OCRを使うことはできるが、特に日本語は満足できる認識率ではないし、数式は一般向けのOCRアプリケーションでは読めない。そもそも画像・図・表など、読み上げることに適さない情報もある。おふとんリーディングは、画像・図・表を認識して切り出す処理も今後実装する予定である。

おふとんリーディングはさらに、今の時代に適した応用が考えられる。CES2014においてウェアラブルデバイスが大量に出展されているが、メガネ型や時計型デバイスに移植すれば、ながら読書に最適だと思われる。幸いAndroidを採用しているデバイスが多いため、操作方法だけ変更すれば簡単に移植できると思われる。

第2章　JavaおよびAndroid開発の学習において特に取り組んだこと

2.1　PDFを表示したい

自炊した本のデータは一般的にPDF形式で保存される。しかしAndroid SDKにはPDFを扱うライブラリは存在しない。そこでMuPDF（http://mupdf.com/）というオープンソースのPDFビューアーを使った。MuPDFはC言語で書かれた軽量かつレンダリングの精細なPDFビューアーである。AndroidやiOSアプリケーションとして配布されているだけでなく、JavaやObjective-Cでラップされ APIを使用することができる。

MuPDFのソースコードをダウンロードしたら、ReadMeに記載されている手順に従って、Android NDKを使ってビルドする。Android用のプロジェクトをEclipseにインポートする。インポートが完了したら、MuPDFのプロジェクトを右クリックし”Properties”を開く。メニューから”Android”を開き、”Is Library”にチェックを入れる。次におふとんリーディングのプロジェクトを右クリックし”Properties”を開く。メニューから”Android”を開き、MuPDFのプロジェクトを”Library”に追加する。これでMuPDFのAPIが使えるようになる。ただし、リソースID（画像や文字列などのID）が重複しているとビルドエラーになる。リソースIDはアプリケーション名などをプレフィックスにして重複しないようにすると良い。

おふとんリーディングは今のところ、MuPDFのAPIのうち、PDFをBitmap（画像のピクセル配列を保持するAndroid SDKのクラス）にレンダリングする機能のみを利用している。文字情報の含まれるPDFは文字を抽出した方が正確である上に処理の無駄をなくすことができるが、今はどのようなPDFでもすべて画像として扱うことによってPDFに関するコードを非常に少なくしている。これをPDFという名前のクラスにまとめた。以下に一部省略して掲載する。

```
public class PDF {
    private MuPDFCore mCore;
    private int pageMax;

    public PDF(Context context, String filePath) {
        try {
            mCore = new MuPDFCore(context, filePath);
            pageMax = mCore.countPages(); // ページ数を利用しなくてもcountPages()は呼ぶ必要がある。
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public Bitmap getBitmap(int page) {
        PointF size = getSize(page);
        // MuPDF1.3では空のBitmapを生成する必要はなく、drawPageの返り値をreturnすれば良い。
        Bitmap bitmap = Bitmap.createBitmap((int)size.x, (int)size.y, Bitmap.Config.ARGB_8888);
        mCore.drawPage(bitmap, page, (int)size.x, (int)size.y, 0, 0, (int)size.x, (int)size.y);
        return bitmap;
    }
```

困ったところがいくつかある。まず、drawPageメソッドは、MuPDF 1.2ではBitmapを引数に渡しているが、1.3では返り値としてBitmapを返すようになった。ここではMuPDF 1.2を使っている。次に、MuPDFCoreのコンストラクタを呼ぶだけでは初期化が完了しなかった。countPagesメソッドを呼ばないと後でエラーが発生する。ソースを目で追っても、ページ番号が-1となりOutOfBoundsExceptionが発生することがわかる。初期化処理がコンストラクタにまとまっていないのはC言語で作られた名残なのだろうと思う。

```
}
```

2.2 文字認識APIに画像をPOSTしたい

PDFをBitmapにレンダリングすることができたら、次は文書のレイアウトを解析したい。つまり文字行を抽出したい。当初は文字行を抽出するための画像認識アルゴリズムを実装するつもりだったが、NTTドコモが文字認識APIをAndroid用のライブラリとして提供していたので利用することにした。ところが、Docomo Developer Supportが開設され、APIの正式版リリースに伴って仕様が大きく変更された。Android用のライブラリが廃止され、HTTPSでPOSTするコードを書くことになった。どうしても正常に動作するコードが書けなかったので、ドコモが主催するWearable Hackathonに参加し、チームメンバーとのペアプログラミングによって解決した。画像をPOSTしてから解析結果を独自のクラスWordに格納するまでの機能をDocomoというクラスにまとめた。ここでは最初の段階である、画像をPOSTするメソッドを掲載する。このメソッドが一人ではどうしても分からなかった部分である。

```
private HttpResponse requestJobID() {
    try {
        DefaultHttpClient client = new DefaultHttpClient();
        // ApiKeyは文字認識APIを利用するためのAPIキーを返すだけのクラスである。
        // GitHubにはこのクラスのひな形である_ApiKeyクラスが置いてあるので、リネームして使う。
        HttpPost post = new HttpPost(
            "https://api.apigw.smt.docomo.ne.jp/characterRecognition/v1/scene?APIKEY=" + ApiKey.getApiKey());
        // マルチパートPOSTするために、MultipartEntityクラスを使うという情報に辿り着くまでに時間がかかった。
        MultipartEntity multipartEntity = new MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
        Fun.cacheImageForDocomo(bmp, 80, bookName);
        FileBody fileBody = new FileBody(new File(Fun.DIR + bookName + "/tmpImageForDocomo.jpg"), “image/jpeg");
        multipartEntity.addPart("image", fileBody);
        post.setEntity(multipartEntity);
        HttpResponse response = client.execute(post);
        return response;
    } catch(Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

新しい文字認識APIはJava用ライブラリを提供しているが、Androidでは動かない。このライブラリは、Apacheのhttpclient-4.2.3.jar, httpcore-4.2.3.jar, httpmime-4.2.3.jarを同梱しているが、これらのメソッドを呼ぶときにNoSuchMethodErrorが発生する。Java用のライブラリがAndroidで使えないことがあるのかと不思議に思っていたが、http://stackoverflow.com/questions/20238421/java-lang-nosuchfielderror-org-apache-http-message-basiclineformatter-instance によると、Android SDKには、Apache HTTP Clientの古いバージョンをベースにしたパッケージが、同じパッケージ名で入っており、手動で追加した4.2.3のjarよりもAndroid SDKに同梱されているjarが優先して呼ばれてしまうようだ。上のコードでは古いバージョンにも存在するメソッドしか使っていないので動くようだ。また、Android SDKに同梱されているApache HTTP Clientは、マルチパート形式の通信機能が省かれているので、httpmime-4.2.3.jarは読み込む必要がある。

2.3 文書のレイアウト解析を自前で実装したい

新しい文字認識APIとの通信は成功したが、これを利用して文書のレイアウトを解析することは困難であることがわかった。古い文字認識APIでは、レイアウト解析のリクエストと文字認識のリクエストが独立しており、レイアウトの解析結果を取得することができた。文字行と認識された領域の上下左右端の座標が返ってくるのである。一方、新しい文字認識APIでは、最終的な文字認識結果のみ取得できるように変更された。文字認識に失敗した領域は取り除かれ、かつ文字認識精度が低いため、ほとんどの領域は取り除かれてしまうことがわかった。

そこで、現在画像認識の実装を行っている。まずはラベリングと呼ばれる処理を実装した。ラベリングとは、2値画像であれば、連続した黒画素の領域に対して、1,2,3…とラベル付けをする処理である。1つの文字は、1つの連続黒画素領域におおよそ対応すると考えられるので、各文字の位置と大きさが取得できると考えた（図表や装飾などをうまく取り除くことができれば）。ラベリング処理は、長尾智晴『図解入門よくわかる最新画像処理アルゴリズムの基本と仕組み』（秀和システム）に掲載せれているアルゴリズムをほとんどそのまま実装した。

ただし、今扱っている画像は32ビットカラーである。2値化することも考えたが、将来の拡張性を考えて、評価関数を渡して判定できるようにした。例えば、輝度にかかわらず黒っぽい画素を文字としてラベリングするために、R,G,Bの値の差が30未満であればtrueを返すというような評価関数を渡すことができる。このような実装は、Java標準ライブラリのCollection.sortメソッドを参考にした。まず以下のような抽象クラスForegroundを定義する。

```
abstract class Foreground { public abstract boolean evaluate(int pixel); }
```

ラベリング処理はLabelクラスにまとめた。Labelクラスのコンストラクタを呼び出すときに、以下のようにBitmapとForegroundクラスを渡すようにする。これは真っ白以外の色であれば全てラベル付けの対象にするという評価関数を渡している。

```
new Label(bitmap, new Foreground() {
    @Override
    public boolean evaluate(int pixel) {
        if(Color.red(pixel) == 255 && Color.green(pixel) == 255 && Color.blue(pixel) == 255) {
            return false;
        }
        return true;
    }
});
```

しかし、ラベリング処理を実行すると、22秒もかかってしまった（1440×2560ピクセルの画像、Galaxy S3で実験）。文字認識APIは回線が十分に早ければ10秒弱で結果を返してくれた。画像をアップロードするよりもローカルで画像処理したほうが速いと思っていたので困った。Android NDKを用いて、C言語で実装したラベリング関数を呼びだそうとしたがコンパイルが通らない。当面は処理速度は度外視して、正しい結果が得られれば良いこととする。

2.4 GUI部品の大きさが0になる

文書のレイアウトが解析できたら、文字行を切り出して電光掲示板のように流したい。右図のように、上半分に文字行を電光掲示板のように流し、下半分にページ全体を画面の横幅に合わせて表示する。

AndroidではGUI部品の大きさを指定するときに、LayoutParamsをセットするという方法を取る。GUI部品のインスタンスに対して、setWidthやsetHeightといったセッタがあるわけではない。そこで、最下層となるmLinearLayoutは画面全体に広げて、電光掲示板のようなmTickerViewと、ページ全体を表示するmPageViewをそれぞれmLinearLayoutの半分の高さになるようにLayoutParamsを設定してみよう。初期化処理だから、onCreateメソッドに書こう。すると画面は真っ白である。mTickerViewとmPageViewの高さが0になってしまうのである。どうしてこういうことが起こるのか。

Androidのライフサイクルを理解するのは難しい。Activityクラスを継承したクラスのライフサイクルについては http://developer.android.com/reference/android/app/Activity.html に紹介されているが、呼び出されるメソッド全てを網羅しているわけではないし、各メソッドで何が行われているかはソースコードを見なければ分からないだろう。

アプリケーションが起動して最初に呼ばれるonCreateメソッドの時点では、子ビューの位置や大きさは確定していない。だから、mLinearLayoutの高さを基準として他の高さを決めようとしてもまだ0が返ってくるだけだ。上のページには書かれていないがonWindowFocusChangedメソッドが呼ばれた時には子ビューの位置や大きさが確定しているらしい。そこで、mTickerViewとmPageViewはonWindowFocusChangedメソッドの中で生成し、LayoutParamsをセットする。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    mLinearLayout = new LinearLayout(this);
    setContentView(mLinearLayout);
    LayoutParams params = mLinearLayout.getLayoutParams();
    params.width = LayoutParams.MATCH_PARENT;
    params.height = LayoutParams.MATCH_PARENT;
    mLinearLayout.setLayoutParams(params);
}

@Override
public void onWindowFocusChanged(boolean hasFocus) {
    super.onWindowFocusChanged(hasFocus);
    if (mRW != mLinearLayout.getWidth() || mRH != mLinearLayout.getHeight()) {
        // mLinearLayoutの幅と高さを取得し、mTickerViewとmPageViewのコンストラクタに渡している。
        mRW = mLinearLayout.getWidth();
        mRH = mLinearLayout.getHeight();
        mTickerView = new TickerView(this, mBook, this, mRW, mRH);
        mPageView = new PageView(this, mBook, mRW, mRH, mBook.getBitmap(mBook.getCurPage()));
        mLinearLayout.setOrientation(LinearLayout.VERTICAL);
        mLinearLayout.addView(mTickerView);
        mLinearLayout.addView(mPageView);
        // LayoutParamsをセットする
        LayoutParams params = mTickerView.getLayoutParams();
        params.width = (int)mRW;
        params.height = (int)mRH / 2;
        mTickerView.setLayoutParams(params);
        params = mPageView.getLayoutParams();
        params.width = (int)mRW;
        params.height = (int)mRH / 2;
        mPageView.setLayoutParams(params);
    }
    isWindowFocusChanged = true;
}
```

よく見ると、mTickerViewやmPageViewのコンストラクタにmLinearLayoutの高さや幅を渡していることがわかる。アクティビティと違い、ビューにはonWindowFocusChangedに相当するメソッドはない。しかし、ビューの中で子ビューを生成した直後はやはりその大きさや位置が確定していないので、基準となる大きさが必要であった。もっと良い方法があるのかもしれないが、今のところ分からない。

LayoutParamsをgetして、その内容を書き換えてからsetするという手順が不思議に感じたかもしれない。当初はこのような書き方はせず、LayoutParamsのコンストラクタを呼び出して生成した後にsetしていた。しかし、ClassCastExceptionが発生する。LayoutParamsは各ビューのクラスごとに存在するのだが、自分はFrameLayoutにはFrameLayoutのLayoutParamsを、LinearLayoutにはLinearLayoutのLayoutParamsをsetするものだと思っていた。しかし、ビューは自分の大きさや位置を決めるために、親のLayoutParamsを受け取っていた。だから型が違うと言われたのである。人間が親を判断してクラス名を書くのは馬鹿馬鹿しいので、getLayoutParamsで親のLayoutParamsを受け取っている。

ビューが自分の大きさや位置を確定するなど、ビューのライフサイクルを理解することは重要だが、これを解説しているページはなかなか見つからなかった。今回はこれらのメソッドをオーバーライドする必要はなかったが、次のページに図解してあるので参考のために掲載する。
http://www.ecoop.net/memo/archives/android_lifecycle_of_view.html

2.5 文書のレイアウト解析が終わったら電光掲示板を表示したい

GUI部品が正常に表示されるようになったら、今度こそ解析結果を使って文字行を電光掲示板のように流したい。文書のレイアウト解析はUIスレッド（メインスレッド）とは別スレッドで行っているため、解析が終了したことを何らかの方法でUIスレッドに伝えなければならない。そこでリスナを定義し、コールバック関数を呼び出して、その中で電光掲示板の表示を開始させよう。以下の様なリスナを定義する。引数にページ番号を渡すことができるようにした。

```
interface RecognizeFinishedListener {
    public abstract void onRecognizeFinished(int recognizingPage);
}
```

文書のレイアウト解析を行うDocomoOldクラスは、BookManagerクラスから呼び出されている。BookManagerクラスは、本のデータを開いたり、現在のページ番号・行番号・レイアウト解析結果などを読み書きするクラスである。Java標準ライブラリのTimerクラスを使って、DocomoOldクラスがレイアウト解析結果を返すのを待ち、上記のリスナを使って、onRecognizeFinished関数を呼び出す。onRecognizeFinished関数の中から電光掲示板を開始させればよいだろう。

```
public void recognize() {
    docomo = new DocomoOld(bmp);
    docomo.start();
    timer = new Timer();
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
            int recognizingPage = mCurPage;
            mPosList = docomo.getWordList();
            if(mPosList != null) {
                timer.cancel();
                Collections.sort(mPosList, new PointComparator());    // 文字行を囲む座標を左上からソートする
                PositionImprove.deleteLongcat(mPosList);       // 誤認識された文字行を削除する
                PositionImprove.deleteDuplicate(mPosList);     // 入れ子になっている・一部重複している文字行を削除する
                mRecognized = true;
                savePageLayout();
                mRecognizeFinishedListener.onRecognizeFinished(recognizingPage); // 最後にリスナに通知する
            }
        }
    }, 0, 1000);
}
```

するとNullPointerExceptionが発生した。既にレイアウト解析が終わっているページについては、解析結果がファイルに保存されているので一瞬でコールバック関数が呼び出されるため、GUI部品の初期化が追いつかないのだ。そこでコールバック関数内でもタイマーを使うことにした。

ところで、Timerクラスを使うためには、わざわざインスタンス変数を用意し、scheduleメソッドの中で、名前なしのTimerTaskクラスを定義し、runメソッドをオーバーライドする必要がある。インデントが深くなって単に見づらいだけでなく、実質的な処理の数とコードの縦の長さが比例しなくなるので気持ち悪い。そこでSimpleTimerというラッパークラスを作ってみる。

リスナを使うことで、コードが縦に長くなることを防ぐ。ただし、タイマーを同じクラス内で複数使いたい場合は、同じコールバック関数内に複数のタイマー終了時処理を書くことになるため、場合分けが簡単にできるよう、任意の文字列を引数messageとして渡せるようにする。またSimpleTimer自身のインスタンスを引数に渡すことで、インスタンス変数を用意することなくcancelメソッドを呼び出すことができる。

```
interface TimerCallbackListener {
    void timerCallback(String message, SimpleTimer timer);
}

public class SimpleTimer {
    private Activity activity;
    private Timer timer;

    public SimpleTimer(Context context) {
        activity = (Activity)context; // ContextをActivityにキャストするのは不思議だが、可能らしい
    }

    public void start(long ms, final String message) {
        timer = new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                // SimpleTimer.thisによって、自分自身のインスタンスをコールバック関数の引数に渡している
                ((TimerCallbackListener)activity).timerCallback(message, SimpleTimer.this);
            }
        }, 0, ms);
    }

    public void cancel() {
        timer.cancel();
    }
}
```

以下のように、ReadingActivityクラス（本を読んでいる時に表示される、おふとんリーディングのメイン画面）において、onRecognizeFinishedメソッド内でSimpleTimerを生成し、timerCallbackメソッド内でタイマーをキャンセルしたあと、電光掲示板を動作開始させている。

```
@Override
public void onRecognizeFinished(int recognizingPage) {
    changeState(StateNormal.getInstance()); // changeStateメソッドは、2.13に関連したものである。
    new SimpleTimer(this).start(100, "onRecognizeFinished");
}

@Override
public void timerCallback(String message, SimpleTimer timer) {
    if (message.equals("onRecognizeFinished")) {
        if(isWindowFocusChanged) { // 2.4において、GUI部品の初期化が終わったら、isWindowFocusChangedフラグをtrueにした
            timer.cancel();
            mTickerView.setImage(mPageView.getImage()); // これで電光掲示板が動作を開始する
        }
    }
}
```

2.6 データを保存したい

ここまでのプログラムによって、本を開いて文字行を電光掲示板のように流すという最低限の動作ができたとしよう（本当は電光掲示板が滅茶苦茶な動作をするのだがその解決は2.9まで待とう）。しかし今の状態では、おふとんリーディングを起動するたびに1ページ目の1行目から再生することしかできない。ページ番号と行番号を保存しよう。

Androidにおいてデータを保存する方法には以下のような方法がある。
(1) SharedPreferencesに保存する
(2) SQLiteに保存する
(3) 任意のファイルに保存する
(1)は数値や文字列などプリミティブな値しか扱えないが、非常に短いコードで書ける。ここでは整数値を保存したいだけなので、(1)が最適のように思える。しかし、(1)は外部ストレージ(SDカード)に保存されない。/data/data/(パッケージ名)/shared_prefs/(パッケージ名)_preferences.xml というファイルに保存される。Android端末を買い替えた時に、SDカードを入れ替えるだけで保存したデータをそのまま使えるようにしたい。Android OS標準のバックアップ機能では、SharedPreferencesのようなデータまではバックアップしてくれないようだ。Android 4.0からは、Android SDKのadbに完全なバックアップ機能が追加されたが、Android SDKの導入は一般の人にとっては敷居が高いのでほとんどの人は使わないだろう。しかもパソコンと接続する必要があるので頻繁に行うのは難しい。(2)も同様に/data/data/(パッケージ名)/database/(任意のデータベース名)に保存されるためバックアップ対象にならない。ここでは、SDカード上にOfutonReadingというディレクトリを作り、その中に書籍ごとのサブディレクトリを作る。その中にテキスト形式でデータを保存しよう。

Android特有の問題について書いておきたい。Androidでは画面サイズやOSのバージョンだけでなく、ディレクトリパスも端末によって異なる。端末間の差を吸収してSDカードのディレクトリパスを取得するメソッドは以下のようになる。これは、 http://inujirushi123.blog.fc2.com/blog-entry-30.html に掲載されている。

```
public static String getExternalStoragePath() {
    String path = null;
    path = System.getenv("EXTERNAL_ALT_STORAGE");
    if (path != null) { return path; }                          // MOTOROLA端末用
    path = System.getenv("EXTERNAL_STORAGE2");
    if (path != null) { return path; }                          // SAMSUNG(新)端末用
    path = System.getenv("EXTERNAL_STORAGE");
    if (path == null) { path = Environment.getExternalStorageDirectory().getAbsolutePath(); }
    File file = new File(path + "/ext_sd");
    if (file.exists()) { return file.getAbsolutePath(); }       // HTC端末用
    return path;                                                // 標準的な方法
}
```

しかし、この方法では不完全であるらしい。http://inujirushi123.blog.fc2.com/blog-entry-93.html によると、/system/etc/vold.fstab にストレージのパスが書かれているので、これを使うのが最良の方法であるらしい（それでも、”Galaxy Nexusや一部のカスタムROMには存在しない”とのことである）。

2.7 Javaオブジェクトをそのまま保存したい

ページ番号や行番号などの整数値は保存できたものの、文書のレイアウト解析結果など複雑なデータはどのように保存すれば良いだろうか。文書のレイアウト解析結果は、1つの文字行ごとにWordというクラスに格納されている。文字行の上下左右端の座標のほか、将来の拡張性を考えて、文字認識APIが返してくれる文字認識結果や認識精度も含んでいる。さらにこれを1ページ分ArrayListにまとめている。このArrayListの中身を、何らかの規則に従ってテキストファイルに保存しても良いが、読み出しが非常に面倒である。

Google GSONというライブラリを使うと、JavaオブジェクトとJSON形式のテキストを簡単に変換することができる。”深い継承階層と広範なジェネリクス型もサポートする”とのこと。TechBooster『Effective Android』（達人出版会）や、 http://gihyo.jp/dev/serial/01/engineer_toolbox/0029 を参考にした。

JavaオブジェクトをJSON形式に変換するためには、Gsonインスタンスを初期化したあと、toJsonメソッドにArrayList(mPosList)とそのクラスを渡すだけである。

```
Gson gson = new Gson();
gson.toJson(mPosList, mPosList.getClass());
```

JSON形式のテキストをJavaオブジェクトに変換するためには、Gsonインスタンスを初期化したあと、fromJsonメソッドにJSON形式のテキストとJavaオブジェクトのクラスを渡すだけであるが、JavaオブジェクトがArrayListなどCollectionの場合は、3行目のような処理をする必要があるらしい。

```
Gson gson = new Gson();
String json = Fun.read(Fun.DIR + mBookName + "/layout/" + fileName); // 保存したJSON形式のテキストを読み込んでいる
Type type = new TypeToken<ArrayList<Word>>(){}.getType();
return gson.fromJson(json, type);
```

2.8 蛍光ペンを引いた部分を一覧表示したい

おふとんで勉強するとはいえ、ただ何もせずに眺めているのでは眠くなるし、復習が困難である。そこで、蛍光ペンを引けるようにして、蛍光ペンを引いた部分をまとめて見られるようにしたい。

蛍光ペンを引く処理は簡単に説明する。GestureDetectorクラスを使ってスワイプ動作を検知する。スワイプした座標を、画面に対する座標系から、文書画像に対する座標系に変換する。Canvasクラスを使って、蛍光ペン用のBitmapに半透明の黄色い長方形を描画し、蛍光ペン用のImageViewに表示する。最後に蛍光ペンを引いた部分を切り出して画像として保存する。画像のファイル名にページ番号や座標を格納する。

蛍光ペンを引いた部分を一覧表示するために、ListViewを使う。ListViewは同じ構造のレイアウトを並べて表示するためのビューである。自分はListViewを使うための手続きが面倒で不自然なものに感じた。だから、こんなことをしなくても、ScrollView上にLinearLayoutを置き、その上に保存した画像ファイルの数だけImageViewを生成しておけばいいのではないかと思った。しかし、そうすると全ての画像がいっぺんに読み込まれることになり、画像の数が増えるとメモリが足りなくなる。ListViewは画面に表示されている部分しかメモリに保持しないようになっているのである。では自分でメモリに読込・解放する処理を書けばいいということになるが、それだけでも大変であるし、ビューを初期化時以外に追加するとなると、2.4のGUI部品を配置するときの悪夢を思い出すのでやめることにした。ListViewの使い方は、次のページを参考にした。
http://techbooster.jpn.org/andriod/ui/1282/

まず、ListViewに並べる1つ1つのアイテムのレイアウトをmarked_list.xmlというxmlに定義した。ほどよく余白を空けてImageViewを配置している。不可視のTextViewを定義しているのは、画像のファイルパスを保持するためである。電光掲示板のように流すときに、Bitmapがほしいのだが、ImageViewからBitmapを取り出すことができなさそうなので、ファイルを読み込むことにした。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp" >
    </ImageView>
    <TextView
        android:id="@+id/fileName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone" >
    </TextView>
</LinearLayout>
```

次に、ListViewに並べる1つ1つのアイテムを格納するデータクラスを定義する。ImageItemという名前にした。画像とファイルパスを保持するだけの単純なクラスである。

```
class ImageItem {
    private Bitmap bitmap_;
    private String fileName_;
    public ImageItem(Bitmap bitmap, String fileName) {
        this.bitmap_ = bitmap;
        fileName_ = fileName;
    }
    public Bitmap getBitmap() {
        return bitmap_;
    }
    public String getFileName() {
        return fileName_;
    }
}
```

そして、このデータをListViewにセットするために、このImateItem型のArrayAdapterを継承したクラスImageAdapterを作る。

```
class ImageAdapter extends ArrayAdapter<ImageItem> {
    private LayoutInflater layoutInflater_;

    public ImageAdapter(Context context, int textViewResourceId, List<ImageItem> objects) {
        super(context, textViewResourceId, objects);
        layoutInflater_ = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // 特定の行(position)のデータを得る
        ImageItem item = (ImageItem)getItem(position);
        // convertViewは使い回しされている可能性があるのでnullの時だけ新しく作る
        if (null == convertView) {
            convertView = layoutInflater_.inflate(R.layout.marked_list, null);
        }
        // ImageItemのデータをViewの各Widgetにセットする
        ImageView imageView;
        imageView = (ImageView)convertView.findViewById(R.id.image);
        imageView.setImageBitmap(item.getBitmap());
        TextView textView;
        textView = (TextView)convertView.findViewById(R.id.fileName);
        textView.setText(item.getFileName());
        return convertView;
    }
}
```

最後に、ListViewを配置しているActivityにおいて、ImageItem型のArrayListに画像とファイルパスを格納し、ArrayAdapterにお願いしてListViewにセットしてもらう処理を書く。

```
File dir = new File(Fun.DIR + bookName + Fun.MARKER);
File[] filelist = dir.listFiles();
try {
    ArrayList<ImageItem> array = new ArrayList<ImageItem>();
    // for文の中で1つずつファイルを読み込み、BitmapとファイルパスをImageItemクラスに格納している。
    for (int i = 0; i < filelist.length; i++) {
        FileInputStream fis = new FileInputStream(filelist[i]);
        Bitmap bmp = BitmapFactory.decodeStream(fis);
        ImageItem item = new ImageItem(bmp, filelist[i].getName());
        array.add(item);
    }
    // ArrayAdapterにお願いしてListViewにセットしてもらう。
    ImageAdapter adapter = new ImageAdapter(this, 0, array);
    listview.setAdapter(adapter);
} catch (Exception e) {
    e.printStackTrace();
}
```

2.9 画面回転・スリープ・バックグラウンド移行に対応したい

これでついにひと通りの機能が実装できた。ほっとして眺めていると端末がスリープしたのでスリープを解除する。すると最初の行から表示されてしまう。

2.4においてActivityのライフサイクルについて書いたが、画面を回転させるとonCreateが呼ばれる。何も考えずに実装していると、画面を回転させただけでアプリケーションを再起動したのと全く同じことになってしまうのだ。なおスリープしただけではonCreateは呼ばれないはずだが、このときはたまたま横画面でスリープ状態に入り、縦向きに持ってスリープを解除したからonCreateが呼ばれたようだった。

さらに、ホームボタンを押してアプリケーションをバックグラウンドに移行させる。しばらく他のアプリケーションを使った後、おふとんリーディングに戻ってみると、かなり先の行まで進んでしまっている。バックグラウンドに移行させるとアプリケーションはほとんど停止した状態になるはずなのになぜだろうか。おそらくアニメーション(ObjectAnimator)はUIスレッドとは別のスレッドで動いているために停止しないのだろう。ホームボタンを押されたことを検知してアニメーションを一時停止する必要がある。Android 4.4 (KitKat)からは、アニメーションを一時停止するメソッドが用意されたが、現時点でAndroid 4.4を搭載しているデバイスは非常に少ないので、ホームボタンが押されたらアニメーションを破棄し、復帰時にアニメーションを生成する必要がある。

まずは、画面の回転に対応させよう。画面を回転した時にonCreateを呼ばないようにする方法もあるが、onWindowFocusChangedが呼ばれなくなり、画面が真っ白になってしまう。2.4の悪夢を思い出すのでやめることにした。行番号を変数に保持するのではなく、行が変わるたびに行番号をファイルに保存することにしよう。すると、onCreateが呼ばれても直前に読んでいた行から始まるので全く問題ない。少し画面の回転は遅くなるが、他のAndroidアプリケーションも、iOSアプリケーションも画面の回転にはずいぶんと時間がかかるものだ。きっと多くのアプリケーションは、これと同様に画面回転時にたくさんの初期化処理をさせているのだろう。

次にバックグラウンド移行に対応させよう。ホームボタンを押したことを検知する機能はないという記事もたくさんあったが、onUserLeaveHintメソッドが呼ばれるということが次のページに掲載されている。
http://y-anz-m.blogspot.jp/2010/10/androidhome.html このように、あるイベントが発生した時に何のメソッドが呼ばれるかはまとめられていないため、ログを出力して自分で調べてみると良いと思う。EclipseのSourceメニューから”Override/Implement Methods…”を開くと、オーバーライドできるメソッドを選んで自動的に追加させることができる。

mTickerViewにアニメーションなどを破棄するdestroyというメソッドを用意し、ReadingActivityのonUserLeaveHintから呼び出すことにしよう。また、アプリケーションの復帰時に、それが復帰なのか起動なのかを区別するために、isPausedというフラグを作り、onUserLeaveHint内でtrueを設定しておこう。

```
public void destroy() {
    if( mAnimatorList.size() > 0 ) {
        for (ObjectAnimator anim : mAnimatorList) {
            anim.removeAllListeners(); // cancelするだけではなくリスナを取り除く必要がある
            anim.cancel();
        }
    }
    removeAllViews(); // アニメーションを止めるだけでなく、ImageViewを取り除く必要がある
    mTickerList = null;
    mAnimatorList = null;
    mTickerList = new LinkedList<ImageView>();
    mAnimatorList = new LinkedList<ObjectAnimator>();
}
```

mAnimatorListはObjectAnimatorを格納したLinkedListである。ObjectAnimatorはローカル変数として生成されたあと、インスタンス変数のLinkedListに格納され、アニメーションが終わると破棄される。ObjectAnimatorをインスタンス変数として生成して使いまわそうとするとおかしな動きをするのでこのようになった。同様の理由で、文字行を切り出した画像を表示しているImageViewもローカル変数として宣言し、LinkedListに格納され、アニメーションが終わると破棄される。

アニメーションはcancelメソッドを呼ぶ前に、removeAllListenersメソッドを呼ばなければならない。cancelメソッドを呼ぶと、onAnimationCancelだけでなく、onAnimationEndも呼ばれるからである。onAnimationEndで次のアニメーションを生成するようにしているので、cancelメソッドを呼ぶだけでは結局次のアニメーションが生成されて止まらないのである。

また、mTickerViewのremoveAllViewsメソッドを呼び、子のImageViewを破棄している。そうしないと、アニメーションが止まってはいても、画像が画面上に残ったままになるからである。

2.10 ファイル選択画面を作りたい

さあ今度こそ完成しただろうか。おや、読みたい本を選択することができないではないか。ファイル選択画面をつくろう。当初は、http://junkcode.aakaka.com/archives/675 を参考にして作っていた。しかし、このプログラムは、最初に開いたディレクトリよりも上の階層に遡ることができない仕様になっていた。これはStackを使っているためである。階層構造を表現する方法としてStackは自然だと思うが、この仕様は不便である。また、自分はファイルを開いた履歴を表示したかったが、このサンプルのようにAlertDialogを使っていると、レイアウトを変更するのが難しいため、1から作ることにした。

自分は正規表現を使ってファイルパスを操作することにした。さらに、ViewGroupを継承したクラスFileListViewを作成し、ListViewを使ってファイル選択画面や履歴を表示することにした。

2.11 画像を表示するとメモリ不足で落ちる

今までは触れなかったが、実はある程度ピクセル数の多い画像を表示すると、アプリケーションが落ちてしまう。アプリケーションに標準で割り当てられているヒープメモリのサイズがかなり小さいそうである。非力な携帯端末で動くアプリケーションであるから、極限までリソースの消費を抑えてほしいというメッセージだろう。しかしそんなことは言っていられないのでヒープメモリを最大限使えるようにしよう。
AndroidManifest.xmlの<application>タグの属性として、

```
android:largeHeap= “true"
```

を追加する。

2.12 なめらかにアニメーションさせたい

2.11と同様に、AndroidManifest.xmlの<application>タグの属性として、

```
android:hardwareAccelerated="true"
```

を追加することはもちろんだが、設定によってはこれだけではアニメーションが滑らかにならないことがある。<uses-sdk>タグの属性に、android:minSdkVersionと、android:targetSdkVersionがある。
targetSdkVersionを最新の値にする。本レポート執筆時点では19(Android 4.4に相当)が最新である。これが例えば、Android 3.0相当の値になっていると、せっかく最新のAndroid 4.4を搭載した端末で実行しても、Android 3.0においてサポートされる機能しか実行されないため、ハードウェア支援などが効かないのである。この値を高くしても互換性には問題がないので常に最新のバージョンを指定しておくと良いと思われる（互換性のために、最も低いバージョンをしているのがandroid:minSdkVersion属性である）。

2.13 プログラムの状態が複雑すぎて手に負えない

何度も修正を繰り返してできあがったプログラムを楽しんでいると、突然クラッシュすることに気づく。そのうち、予期していなかったユーザー入力がたくさんあることに思い至る。例えば、文書のレイアウト解析が終わっていない状態で、さらに次のページに進むとか、蛍光ペンを引こうとするとか、他にも様々な操作でクラッシュすることがわかる。しかしもうプログラムは膨大な行数になっており、あちこちにエラー回避のためにif文を仕込んだつもりである。あらゆる状態に対して動作を場合分けしようとしても、もう頭がついていけなくなっている。

Stateパターンというデザインパターンが、http://codezine.jp/article/detail/5957 のページで詳細に解説されていた。デザインパターンという言葉は聞いたことがあったが、言語の文法もあやふやであった自分には遠い存在に感じてどういうものであるかさえ知ろうとしなかった。デザインパターンは、「こういう種類のプログラムを書くなら、こういう風に書くとごちゃごちゃせずにメンテナンスしやすいコードが書けるよ」という書き方のことであった。結城浩『Java言語で学ぶリファクタリング入門』（ソフトバンククリエイティブ）も非常に参考になった。

Stateパターンでは、プログラムの状態を表すためのクラスを作り、そのクラスを切り替えることで条件分岐を実現する。Stateパターンのメリットは、if文などが消えることでスッキリしたコードになると解説されていることもあるが、それよりも、分岐がいったいいくつあるのかを頭のなかで整理できることが大きなメリットであると思われる。例えば、プログラムの状態が4種類あるとしよう（通常の状態・文書のレイアウト解析中・アニメーション一時停止中・エラー発生中）。さらに、ユーザ入力が5種類あるとしよう（ページをめくる・行を移動する・一時停止または再開させる・スリープする・蛍光ペンを引いた部分の一覧を開く）。すると、ユーザー入力が発生した時に、何か特別に処理をしなければいけない可能性が4×5=20通り存在することがわかる。少しでも実用的なアプリケーションであれば、すぐにこのくらい複雑になってしまうだろう。Stateパターンを使えば、この20通りのコールバック関数を作り、ひとつひとつ処理を実装すれば良い。もしif文を散りばめて20通りの処理を網羅しようとしたら発狂してしまうだろう。

第3章　おわりに

3.1　本演習の感想

本演習を通して大量の本とWebページを読み、おふとんに入らなくても本が読めるような頭を手に入れた。さらに、携帯端末に対する物欲がかなり弱くなった。アプリケーションを開発できるようになったことで、携帯端末の独自機能や性能に依存せずにすむようになったのだ。今後iOSやMac OS X、WindowsやLinuxのアプリケーションを開発できるようにすれば、さらに世の中の商売から自由になれるだろう。

それはさておき、本演習を終えてようやくプログラムを書いた実感を得ることができた。大学に入学してプログラミングを始めて2年近く経つが、やっと自分は実用的なプログラムを書くことができるかもしれないと少し自信を持つことができた。自分の書いたソースの規模ももちろんだが、Android SDKの膨大なクラス群、そしてそれを理解するためのドキュメントの規模がものすごく大きく、これを多少なりとも使えるようになったことが自信を与えてくれる。しかも前半に紆余曲折があったため、JavaやAndroid開発の勉強ができたのは約4ヶ月である。短い期間で多くの勉強をする方法を身につけることができたのではないかと思う。

3.2　ソースコードとライセンス

おふとんリーディングのソースコードは、https://github.com/ralit/OfutonReading に公開されている。MuPDFがGPLv3ライセンスのもとに公開されていることから、おふとんリーディングもGPLv3ライセンスのもとで公開する。

ソースコードをビルドするときは、Docomo Developers Supportに登録し、文字認識APIのAPIキーを手に入れ、_ApiKeyクラスをApiKeyにリネームし、APIキーを貼り付ける。

MuPDFのソースは同梱していないので、別途ダウンロードし、platform/android/ReadMe.txtの手順に従いndk-buildする。本レポート2.1を参考にして、Eclipse上で簡単にライブラリとしてリンクさせることができる。

付録　主に情報メディア特別演習の受講を検討している学生に向けて

付録1 膨大なSDKを前にして絶望したとき

http://developer.android.com/reference/packages.html を見てみよう。膨大なクラス群に圧倒される。APIリファレンスはどこから手をつけていいのかわからないから、今度はチュートリアルを読んでみようとして、http://developer.android.com/guide/index.html を開く。しかし分量が多くて、「これをじっくり読もうとして眠くなってほとんど進まず寝るということを毎日繰り返すと、今までの経験上なにもできずに1年経ってしまう」と思うとじっくり読もうという集中力も起こらない。APIリファレンスに戻ってみたり、「なにか自分のやりたいことに似ていて、しかもそのままコピペで動きそうなコードを掲載しているブログ記事はないかな？」と検索してみたりする。「おふとんリーディングを作るなら、まずは画像を表示する必要があるな」と考えて、画像に関係がありそうなクラスをAPIリファレンスから探してみる。「Bitmapというクラスがあるな。BitmapFactoryって一体何が違うんだろう。Canvasというのも画像っぽい気がするな。Paint？Picture？一体似たようなクラスがいくつあるんだ。どこから手をつければいいんだ？」

サンプルコードを読んでも、どうも理解したような気分になれない。Javaの知識がないのに、Javaを勉強せずにコードを読もうとしていたのだ。時間もないし目の前に今すぐ役立つかもしれないコードがあるのに、ゆっくりJavaの文法を勉強していたらAndroid開発がいつまでたっても始められない気がした。というのも、自分は独学が苦手なようで、自分で何かプログラミング言語を勉強しようとしても、かなり長い期間がかかった挙句途中で放棄し、少し読んだはずの文法もすぐに忘れてしまったという経験があるからだ。そこまで焦っていたのは、Androidで開発しようと決めたのが8月末だからである。それまでの紆余曲折は付録2に書くが、Qtでの開発が今までの経験通り、お勉強の段階で行き詰まって一向にアプリケーションづくりが始まらなかった。8月31日に先生とミーティングがあったのだが、ミーティングが数日後に迫ってくると、「Qtは諦めて、Androidで開発した方がいいのではないか」と思った。その時に買った本が、高見知英『よくわかるAndroidアプリ開発の教科書　Android 4.2 対応版』（マイナビ）である。この本は発売日に買ったもので当然最新の開発環境に対応しているから環境構築の時点でつまづくおそれがない。しかも非常に良く出来ている本であった。初心者に優しく、下手に省略がないのでつまづくおそれがない一方、内容は非常に充実していて、音声認識機能やGoogleMapsなど見た目の派手な内容がたくさん盛りこんであり、飽きずに読むことができる。マルチスレッドなど高度な内容も含んでおり、12月になっても読み返すことが多かった。

8月31日のミーティングにおいて、Android SDKで開発することに変更し、進捗報告として、上記の本を読んだり写経したりしたことを報告した。先生にはJavaの文法が怪しいと指摘され、三谷純『Java 1 はじめてみようプログラミング』『Java 2 アプリケーションづくりの初歩』（翔泳社）を勧められた。正直、古い本だし、自分の大学の先生が書いた本がそんなにすごいとも思えなかったのだが、帰りに図書館に寄ってその晩に第1巻を読んだ（次の晩に第2巻を読んだと書きたいところだが、全身に蕁麻疹が出て一晩中眠れないし何をしていても辛かった。しばらく不眠のような日が続いた）。第2巻も数日で読めた。これは、ミーティングをした直後の緊張感をうまく利用できたことも大きいだろう。しかし、この本が目的にかなっていたことも大きいと思う。その目的とは、まったくの初心者が時間をかけずにまとまった知識を習得することである。この本は見た目には平凡な本なのだが、自然と次々にページをめくってしまう。行間や余白が大きいためだろうか。ページ数は多くないが、情報量が不足しているとは感じない。最近、この本を人に勧めたが、同じような感想を持ったようだ。このような本を探し当てるには、「どんどん読めそう」という感覚が大事なので、大きな本屋さんに行って、片っ端からページをパラパラめくって、「なんだか知らないけどこの本はどんどん進めそうだ」と感じる本を探し当てるのがいいだろう。普段はAmazonのレビューに頼っているが、Amazonのレビューはこのような観点から評価していることは少ないので今回は参考にならない。総当りが必要だ。

プログラミング言語の文法を数日で勉強し、ターゲットとするプラットフォームの勉強（自分の場合は上記の入門書を写経した）を数日行い、合計一週間くらい経ったら、「実現したい機能の断片を、2,3行の短いコードを試しに書いて実行しては消す」という作業を繰り返すのが理想的ではないだろうか。ターゲットとするプラットフォームの入門書を読み通せば、「実現したいあの機能はこれを使えばできそうな気がする」ということがたくさん思いつくと思う。もちろん、知識が足りないので実装しようとするとすぐに行き詰まって、検索しまくることになる。すぐに行き詰まることが分かりきっているので、コードを書くこと自体が億劫になって、「しばらくは知識をつけるために本を読んだり検索したりすることに専念しよう」と思うのだが、やろうとしている事自体が複雑なため、知識をつけようとしてもそれが頭の中で整理できず、結局は実装するときに同じようなことを1から検索し直すことになる。とはいえ、いきなりアプリケーションを構築しようとするのは難しいので、2,3行の断片を書くのである。例えば「（リソースからでいいから）画像をImageViewに表示する」「（画像じゃなくていいから）なにかファイルから読み込む」など、大きく妥協して2,3行で書けるコードを書いて実行してみることを繰り返す。幸い、Javaは、開発環境が整っているので、実行する前からコンパイルが通るかどうかが分かるし、メソッドが自動補完できる（説明も表示される）ので、ブラウザを開いてAPIリファレンスを読まなくても、メソッド名から推測してそれっぽいものを片っ端から試すことが楽に出来た。

もし自信がないなら、できるだけ開発人口が多いターゲット、特に最新環境の開発人口（できれば日本人）も多いターゲットを選ぶと良いと思う。その点でAndroidは恵まれていた。やりたいことはどこかのブログに見つかるし、エラーメッセージで検索すれば、ほとんど全てのエラーが　http://stackoverflow.com/ で解決済みであった。iOSもきっとそんな感じだろう。WindowsやMac OS Xも人口は多そうだが、モバイルOSと比べると勢いがないので最新の記事は少ないかもしれない。Qtは特にQt 5の開発人口が少ないと感じた。日本語の情報となると更に少ない。

他に参考にした本は、Paul Deitelら『プログラマーのためのAndroid アプリ駆動アプローチ』（ピアソン桐原）、出村成和『Android NDK ネイティブプログラミング 第2版』（秀和システム）だが、この2冊はあまり読んでいない。Androidの基礎は前ページに挙げた本で十分だと思うし、Android NDKに関しては、自分でC言語のソースを書かない限りこの本の内容まで学ぶことは必須ではない。ただ、オープンソースのCライブラリをビルドする方法などが書かれている（第1版には書かれていないので注意）ので、必要になった時に安心だろうと思って買った。MuPDFはEclipseプロジェクトが同梱されているので、この本の知識は必要なかった。

付録2 演習テーマを2度変更したことについて

付録1に書いたように、8月末まではQtを使っておふとんリーディングをマルチプラットフォーム開発しようとしていた。QtはC++のライブラリで（ライブラリと言っても、マクロを使ってC++言語にはない書き方をさせるなどアグレッシブなライブラリである）、Linux, Mac OS X, Windows 向けのアプリケーションを同時に開発できる。Qt 5.1でAndroidに対応し、Qt 5.2でiOSに対応した。Adobe製品もQtを使って開発されていると聞いて（現行バージョンがQtを使って開発されているかどうかは分からない）、将来性のあるツールだと思ってQtを選んだ。Qt 5は正式版が2012年12月に発表された新しいものであり、開発が活発に行われていそうなところも気に入った。

しかし、QtはネイティブのAndroidやiOSの開発者と比べると圧倒的に人口が少ない。今ならAndroid SDKを使った開発経験を通して、Qtの開発を勉強できるかもしれないが、何も経験がない当時は、優しく書かれた、しかもそのまま動くコードが載っているブログ記事が少ないとすぐに行き詰まってしまった。日本語情報以外も含めればそれなりの情報があったかもしれないが、当時の自分が探した時の印象では少ないと思った。公式のチュートリアルもAndroidやiOSと比べれば充実していない。

最初に読んだ本は、折戸孝行『Qt Quickではじめるクロスプラットフォーム UIプログラミング』（アスキー・メディアワークス）である。これも、Androidの入門書と同じく、発売日に買った記憶がある。これはQt 5から導入されたQMLというQtの独自マークアップ言語を中心にしてアプリケーションをつくろうという本である。日本で初めてQt 5に対応した本であると思われる。省略が少なく、飽きずに次々にページがめくれるようなレイアウトで、初心者に優しい本である。QMLとC++を組み合わせる書き方についても書かれているが、ほとんどすべてがQMLだけで書くことを想定しているので、この本だけでQtの全貌を理解することは難しかった。

次に読んだ本は、Jasmin Blanchetteら『入門 Qt 4 プログラミング』（オライリー・ジャパン）である。こちらはQMLが登場する前の従来のQtの書き方を勉強することができる。ただ、非常に内容が詰まっているので最初のほうしか読んでいない。津田伸秀『Qtプログラミング入門』（工学社）が同じくQt 4の本で、パラパラと読めそうに見えて買ったのだが、なんだかわかりにくくて読むのをやめてしまった。

Hello WorldをAndroid向けにコンパイルする段階で行き詰まってしまい、挫折した。今はどうなっているかわからないが、当時はクロスビルドを全自動で行えなかった。Mac用にビルドするなら、.appファイルを生成した後に一部のライブラリを.appファイル内にコピーする必要があった。コンパイルやリンクについては全く知識がなかったので、飯尾淳『C言語による スーパーLinuxプログラミング』（ソフトバンククリエイティブ）の3,4,5章を読んだ。しかし今でもコンパイルやリンクは理解できていない。

また、当時は文書のレイアウト解析を自前で実装するつもりでいたことから、昌達慶仁『C言語で実装する画像処理アルゴリズムの全て [詳解]画像処理プログラミング』（ソフトバンククリエイティブ）の使えそうな部分などを読んだ。

演習テーマを「2度」変更したとはどういうことかというと、情報メディア特別演習を開始した当初は、本を読むのではなく、本をノートにまとめることを支援するWebアプリケーションの開発を考えていた。プレゼンテーション資料のような形式で簡単にノートまとめを行なえて、その資料を自動再生するWebサービスである。しかし、そもそも何らかの本1冊を読みながら、プレゼンテーション資料のようなノートまとめをすること自体が現実的ではない。そんなに時間のかかる勉強法は絶対にできないと思った。自分さえ使えないサービスだと思った。またJavaScriptでインタラクティブなUIを書くことが大変だった。しかし、楽に勉強したいという方向性はブレていないことが分かる。

付録3 Docomo Wearable Hackathonの報告

2.2で述べたように、NTTドコモ新しいAPIと正常に通信することができなかったので、Docomo APIを使ってアプリケーションを開発しようというこのハッカソンに参加し、チームの方とペアプログラミングを行って動くコードを書くことができたのであった。ハッカソンで作ったアプリケーションについて述べたい。なお、ハッカソン全体の様子については以下の記事で詳細に報告されている。
http://devlog.dcm-gate.com/2013/12/wearable-hackathon.html

このハッカソンは、Docomoの提供するAPIを使うことと、ウェアラブルデバイス（メガネ型デバイスなど）を使うことを条件としたものである。2013年12月7日と8日に行われ、開発時間は約12時間であった。自分の所属するチームは「ミミミル」という視覚障害者向けのアプリケーションを開発した。目で見えないものを代わりに耳で見るという意味である。使用したデバイスはVuzix M100というメガネ型デバイスで、使用したAPIは、文字認識API、音声認識APIである。ミミミルはまだ一般公開されていないが、障害者用施設などで実証実験を行う準備が進められている。また、次のWebページも用意されている。http://mimimiru.jp/

ミミミルはプログラムの力と人間の力を使って視覚障害者を補助するアプリケーションである。人間の参加が欠けていても完結できる点も特徴である。例えば目の前にあるものが何であるかを知りたいとき（似たようなボトルが並んでいるが片方が飲み物で片方が調味料であるという場面を想像してみよう）、メガネ型デバイスのボタンを押すと、写真が撮影されると同時に音声入力が開始される（例えば「どっちが飲み物ですか」などとしゃべる）。すると音声入力結果と写真がTwitterに投稿される。誰かがその投稿を見て答えをリプライしてくれると、ミミミルはリプライを読み上げる。一方、音声入力をしている間に、写真は文字認識APIにも送られている。文字認識結果が返ってくると、ミミミルは文字認識結果を読み上げる。人力の遅さを解消するためにプログラムの力も使うわけである。文字は左上から順番に読み上げられるため、文字認識がうまく行けば、どちらが飲み物であるかわかるかもしれない。

ミミミルの開発において、自分は文字認識APIとの通信、Twitterへの投稿・リプライの取得、ソースコードのマージを担当した。文字認識APIは前述のとおりチームメンバーとのペアプログラミングによって解決した。Twitterへの投稿・リプライの取得は、やはりチームメンバーとのペアプログラミングで、ペアの方が以前作成したTwitter関連のプログラムから切り貼りして作った。Twitterに関する処理は、発表まで2,3時間しかないところ、急遽担当して急いで作ったため、理解する余裕がほとんどなかったが。ソースコードのマージも急遽担当したのだが、その2週間前に、ほとんど全てをMainActivityに詰め込んでいたおふとんリーディングの汚いソースコードを、オブジェクト指向にしたがって書きなおした経験があったため、力になることができた。また、チームメンバーは、アプリケーションのMainActivity、写真撮影、音声認識APIとの通信、Webサーバ側の実装、Webページの作成、プレゼンテーション資料の作成などを行った。参加者とメンターの投票により自分たちのチームが優勝した。

このようなイベントは結構頻繁に行われているようなので参加することをおすすめしたい。特に今回のように大手企業が主催しているイベントは、教えてくれる人がいるので、初心者でも参加しようという気持ちになることができた。JavaとAndroidの経験が3ヶ月半くらいであったが、Javaの文法をひと通りおさえ、ブログなどに載っているコードを読めば意味が理解できる状態になっていれば、かなり対等な立場で開発に参加できることがわかった。