

平成 25 年度「成長分野等における中核的専門人材養成の戦略的推進」

Android アプリケーション開発教材

スマホアプリ開発技術者育成のためのカリキュラム・教材開発と評価指標検証

概要

目的

- Android アプリケーション開発に必要な基本的なプログラミング知識を身につける。
- 演習を通して、アプリケーション開発を体験する

受講前提条件

- Java の基本的なプログラミングスキル
- オブジェクト指向の基本的な知識

開発環境

PC の開発環境

本書で使用している Android 開発環境は以下のとおりです。
この章では、Android SDK、 SDK Platform をインストールします。

表 1 開発環境

OS	Windows 7
作業ディレクトリ	c:¥android_training_basic
Java SDK	JDK 1.6
AndroidSDK	Ver 22.3
SDK Platform	Android 4.4 (API 19)

- <注意>
 - c:¥android_training_basic フォルダは各自で作成すること
 - Java は事前にインストールされていること

付属 DVD について

演習に必要な全てのツールは、本書付属の DVD で提供済です。
DVD の中に「android_training_basic.zip」というファイルが1つ用意されています。

zip ファイルを解凍すると「android_trainig_basic」フォルダが作成され、その中には表 .2 ようなファイル及びフォルダが用意されています。

表 2 android_trainig_basic.zip の中身

ディレクトリ名	説明
adt-bundle-windows-x86_64-20131030.zip	Android 開発ツールと Eclipse(64bit)
adt-bundle-windows-x86-20131030.zip	Android 開発ツールと Eclipse(32bit)
workspace	Eclipse のワークスペース
answer_docs\html	実習の解答ドキュメント

インストールするツール

本書では、表 .3 のツールをインストールします。

表 3 開発環境

ソフトウェア	バージョン
Eclipse	Eclipse IDE with built-in ADT
AndroidSDK	Ver 22.3
SDK Platform SDK	Android 4.4 (API 19)

ツールの詳細については「第 3 章 開発環境の構築」で説明します。

演習の進め方

解答ドキュメントの開きかた

実習の解答は別ドキュメントとして用意されています。次の手順で解答ドキュメントを開きます。

手順

1. 付属 DVD android_trainig_basic.zip を解凍する
2. 生成されたフォルダ android_training_basic を開き answer_docs¥html に移動する
3. index.html を開く
4. ブラウザが起動し、解答ドキュメントのページが表示される

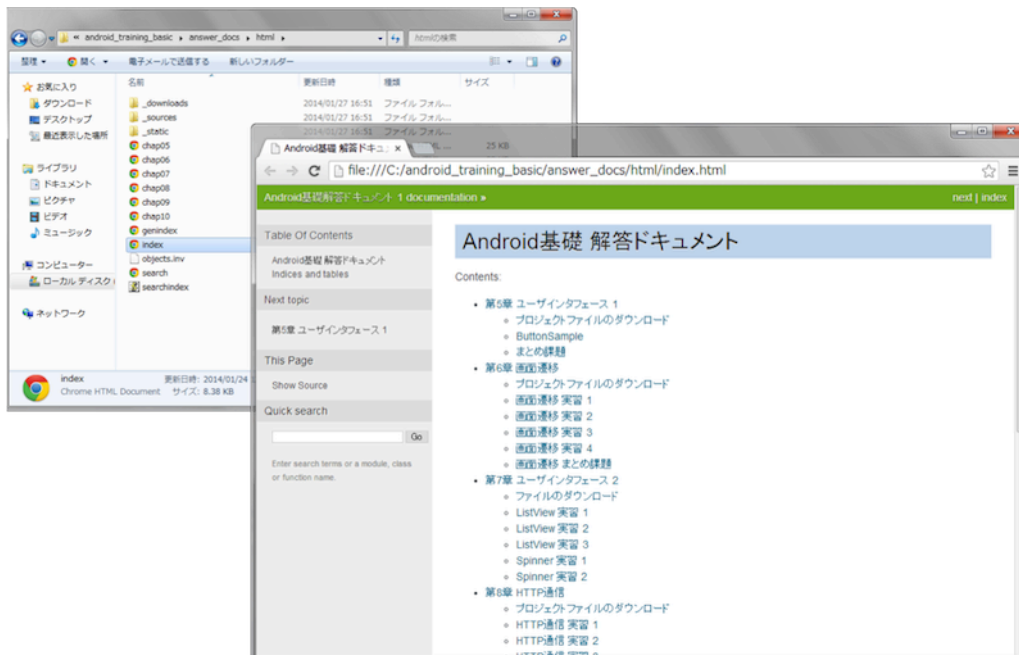


図 1 解答ドキュメント

実習用プロジェクトのダウンロード

実習用プロジェクトは解答ドキュメントから取得できます。
各章ごとに完成プロジェクトと実習に必要なスケルトンプロジェクトを含んだプロジェクトファイルが zip 形式で用意しています。

手順

1. answer_docs/html/index.html を開く
2. 取得対象の「プロジェクトファイルのダウンロード」リンクをクリックする
3. [ダウンロード] をクリックすると、対象ファイルのダウンロードが開始される

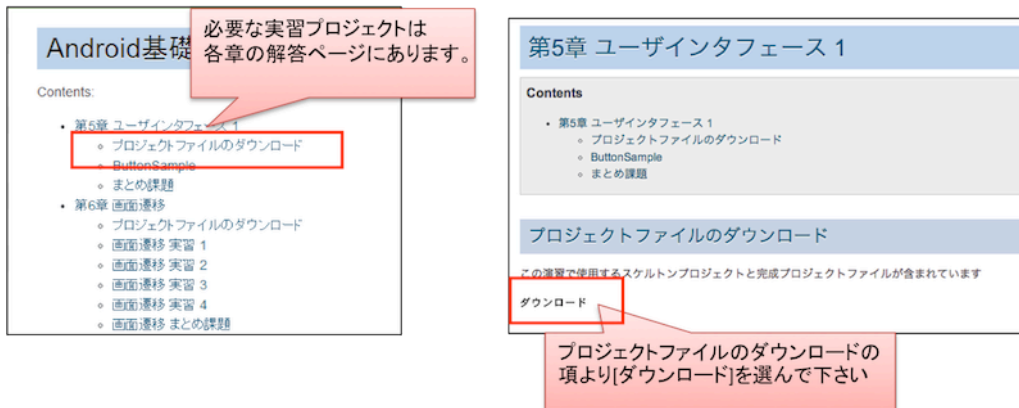


図 2 実習用プロジェクトのダウンロード

プロジェクトのインポート

トレーニング実習ではあらかじめ用意してある、スケルトンプロジェクトを使って実習をすることがあります。

プロジェクトのインポートは次の手順で行います

手順

1. Eclipse の [File] メニューから [Import] を選択する
2. [Import] 画面で、[General] > [Existing Projects into Workspace] を選択し、[Next] をクリックする
3. 次の画面で [Select archive file] にチェックをいれ、[Browse] ボタンをクリックする
4. 解答ドキュメントよりダウンロードした zip ファイルを選択する
5. Import 一覧が表示されるので、目的のプロジェクトにチェックを入れ、[Finish] ボタンをクリックする

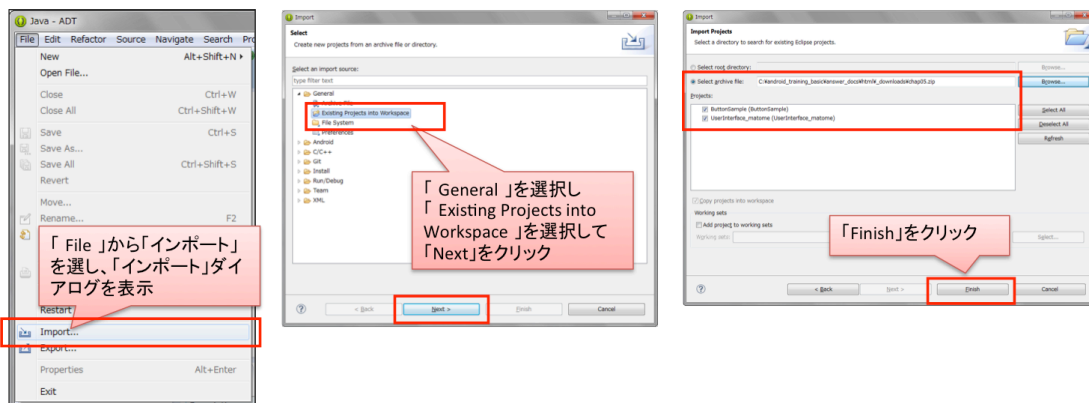


図 3 プロジェクトのインポート

Task ビューの表示方法

スケルトンプロジェクトを使った実習では、修正箇所をタスクに登録しています Task ビューを使うと登録されているタスクの一覧を表示できます。

次の手順で Task ビューを表示させます。

手順

1. Eclipse の [Window] メニューから [Show View] > [Tasks] を選択する
2. Eclipse にタスク一覧が表示される

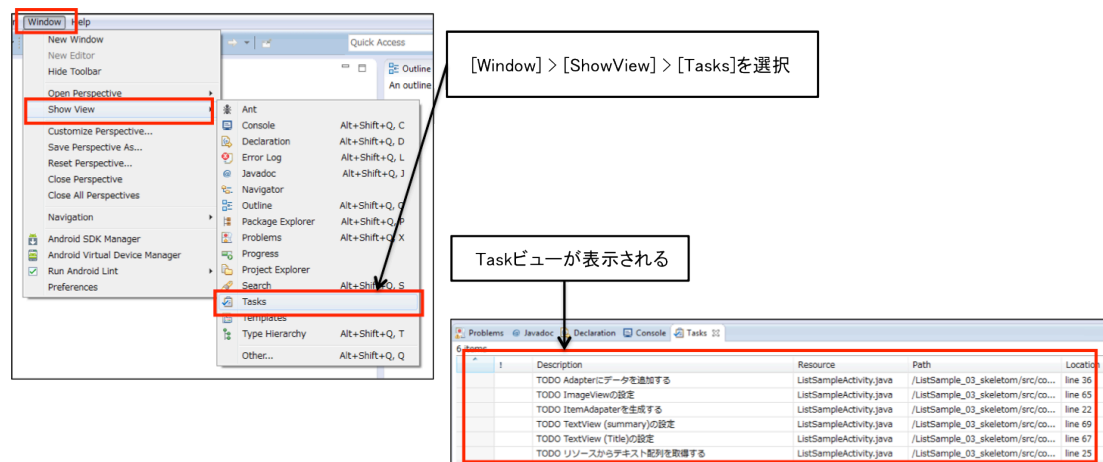


図 4 Task ビューの表示方法

目次

概要	i
目的	i
受講前提条件	i
開発環境	i
演習の進め方	ii
第1章 Android とは	1
1.1 Android とは	1
1.2 Android のバージョン	3
第2章 Android のアーキテクチャとコンポーネント	7
2.1 Android のアーキテクチャ	7
2.2 アプリケーションコンポーネント	10
2.3 ライフサイクル	14
第3章 開発環境の構築	17
3.1 Android アプリケーションの開発環境を準備する	17
3.2 開発環境とインストールツールについて	17
3.3 開発環境の構築	19
3.4 エミュレータの作成	27
第4章 開発ツールの使い方	33
4.1 アプリケーションの作成	33
4.2 Activity	38
4.3 リソースファイル	39
4.4 画面デザインを変更する	40
4.5 View の整列	43
4.6 EditText の追加	45
4.7 文字列リソース	47
4.8 AndroidManifest.xml	50
4.9 アプリケーションログの参照	52
第5章 ユーザインタフェース 1	55

目次

5.1	View	55
5.2	View の作成	57
5.3	クリックイベント	67
5.4	【実習】Button	73
5.5	ViewGroup	83
5.6	OptionsMenu	93
5.7	Toast	95
5.8	AlertDialog	97
5.9	まとめ課題	100
第 6 章	画面遷移	102
6.1	シンプルな画面遷移	102
6.2	【実習】画面遷移 1	105
6.3	遷移元の画面に戻る	112
6.4	【実習】画面遷移 2	113
6.5	画面遷移の連携	115
6.6	【実習】画面遷移 3	118
6.7	遷移先画面から結果を受け取る	121
6.8	【実習】画面遷移 4	124
6.9	まとめ課題	127
第 7 章	ユーザインタフェース 2	129
7.1	List View	129
7.2	【実習】List View 1	135
7.3	一覧のアイテムを選択する	139
7.4	【実習】List View 2	140
7.5	List View のカスタマイズ	142
7.6	【実習】List View 3	147
7.7	Spinner	150
7.8	【実習】Spinner 1	155
7.9	Spinner を選択する	159
7.10	【実習】Spinner 2	161
第 8 章	HTTP 通信	163
8.1	Web サービスに接続する	163
8.2	【実習】HTTP 通信 1	167
8.3	レスポンスデータから必要な情報を取得する	171
8.4	【実習】HTTP 通信 2	172
8.5	WebAPI	174
8.6	【実習】HTTP 通信 3	178

第9章	JSON 解析	182
9.1	JSON	182
9.2	【実習】JSON	186
第10章	データベース	193
10.1	SQLite	193
10.2	【実習】データベース 1	200
10.3	sqlite3	205
10.4	【実習】データベース 2	207
10.5	データの検索	210
10.6	【実習】データベース 3	212
10.7	データの追加	217
10.8	【実習】データベース 4	218
10.9	レコードの内容を取得する	224
10.10	【実習】データベース 5	226
10.11	データを一覧表示する	228
10.12	【実習】データベース 6	230
10.13	条件検索	234
10.14	【実習】データベース 7	235
10.15	データの更新	240
10.16	【実習】データベース 8	242
10.17	データの削除	247
10.18	【実習】データベース 9	248

第 1 章

Android とは

この章の目的

- Android とは何かを理解する

1.1 Android とは

Android とは何か

Android は、スマートフォンやタブレットなどの携帯端末向けの OS です。実際には、ライブラリ、Android ランタイム、主要アプリケーションなどもまとめて提供されています。

Android アプリケーションの開発を行うために必要な Android SDK (Software Development Kit) は、無償でダウンロードして利用できます。また、Android はオープンソースプロジェクトであるため、ソースコードが公開されています。

Android アプリケーションは Java で開発します。開発したアプリケーションは、Google Play で公開して販売することができます。



図: ハンドセット、タブレット

Android が普及した経緯

2007年11月、Googleを中心としてOHA（Open Handset Alliance）というコンソーシアムが結成され、携帯端末向けプラットフォーム、Androidの開発を推進することが発表されました。

2008年10月、Android 1.0を搭載した最初のAndroid端末（T-Mobile G1）がリリースされました。2009年2月には、同機種用のアップデートとしてAndroid 1.1がリリースされ、同年4月にAndroid 1.5、9月に1.6、10月に2.0、.....とバージョンアップを重ねていきます。

同時に、Android搭載スマートフォンが市場でのシェアを広げていきました。スマートフォンの世界市場では2010年第4四半期以降1位を占めています。

AndroidはLinuxをベースとしたOSであり、オープンソースであることから、携帯端末に限らずさまざまな機器に移植できます。2012年には、Android搭載のスマートフォンと連動する家電が登場しました。今後もさまざまな分野で活用されていくことでしょう。

ハンドセット、タブレット以外の Android

- ノート PC
- MediaController
- TV
- 腕時計



図 1.1 ハンドセット、タブレット以外の Android

Android OS を搭載した組み込み機器

- カーナビ
- 洗濯機
- 電子レンジ
- FAX



図 1.2 Android 組み込みハードウェア

1.2 Android のバージョン**Android のバージョンと新機能**

Android では、バージョンアップとともに様々な新機能をサポートしてきました。頻繁なバージョンアップがユーザーに高機能な端末をいち早くリリースすることを可能にしてきました。また、同時に、アプリケーション開発者にとってはアプリケーションのメンテナンス頻度が上がるため、作業が増えてしまう原因にもなっています。

Version	コードネーム	APIレベル	リリース	新機能
2.0/2.1	Eclair	5, 6, 7	2009/10/26 (2.0) 2009/12/3 (2.0.1) 2010/1/12 (2.1)	<ul style="list-style-type: none"> マルチタッチ LiveWallPaper Bluetooth
2.2	Froyo	8	2010/5/21	<ul style="list-style-type: none"> Dalvik VM にJITコンパイラを搭載 (2~5倍高速化) クラウドとデバイスの連携API (C2DM) テザリング対応 Adobe Flash対応 インストール済アプリの自動更新
2.3	Gingerbread	9, 10	2010/12/6	<ul style="list-style-type: none"> ゲームのための改良 並列GC (目標3ms以下の停止) NFC (近距離無線通信) 対応 複数のカメラを扱えるAPIの追加 SIPの標準サポート バッテリー管理機能の向上

図 1.3 Android 2.x 系の新機能

Version	コードネーム	APIレベル	リリース	新機能
3.0	Honeycomb	11	2011/2/22	<ul style="list-style-type: none"> 大型ディスプレイに最適化 タブレット専用となった マルチコアプロセッサのサポート
3.1	Honeycomb	12	2011/5/10	<ul style="list-style-type: none"> ユーザーインターフェースの改善 オープンアクセサリAPI USBホストAPI マウス、ゲームパッド、ジョイスティックからの入力 ホームスクリーンウィジェットのサイズ変更
3.2	Honeycomb	13	2011/7/15	<ul style="list-style-type: none"> 広範囲なタブレット向けの最適化 SDカードに対してのメディア同期 スクリーンサポートの拡張

図 1.4 Android 3.x 系の新機能

Version	コードネーム	APIレベル	リリース	新機能
4.0	Ice Cream Sandwich	14, 15	2011/10/08	<ul style="list-style-type: none"> • ハンドセットとタブレットのUIの統合 • Android Beam • WiFi Direct • Bluetooth Health Device Profile • Notificationの向上 • ロック画面で、カメラと音楽の操作 • ランチャーのアプリ管理の改善 • 画像や動画のエフェクト • 正確なカメラの測光、顔認識
4.1	Jelly Bean	16	2012/06/27	<ul style="list-style-type: none"> • Systrace • アクセシビリティの拡張 • 双方向テキスト対応 • Unicode 6.0 の絵文字対応 • Notificationの拡張 • リサイズ可能なアプリウィジェット • ライトアウト・フルスクリーンモードへの遷移 API • Remoteable View の追加 • デバイスの追加と除去の検知 • Android Beam の改善

図 1.5 Android 4.0, 4.1 の新機能

Version	コードネーム	APIレベル	リリース	新機能
4.2	Jelly Bean	17	2012/11/13	<ul style="list-style-type: none"> • マルチアカウント • クイック設定 • フォトギャラリーのアップデート • 360度撮影 • ジェスチャ文字入力 • Google Play 以外からインストールするアプリにもマルウェアスキャン
4.3	JellyBean	18	2013/7/24	<ul style="list-style-type: none"> • 描画処理の高速化 • OpenGL 3.0対応 • バッテリー節約 • SELinux対応 • Wi-Fiに接続することなく、Wi-Fi による位置情報取得
4.4	Kitkat	19	2013/10/31	<ul style="list-style-type: none"> • ART仮想マシンの導入 • 512MB DRAM対応 • NFC が Host Card Emulation 対応 • 印刷フレームワーク • Chromium WebView • メモリ使用量解析

図 1.6 Android 4.2 以降の新機能

Android のバージョンと API レベル

Android の各バージョンには、API (Application Programming Interface) Level が割り当てられています。API Level とは、Android が提供するフレームワーク API を識別する整数値のことです。Android アプリケーションでは、バージョンではなく API Level で設定を行う場合があるため、注意してください。

Android のバージョンと API Level は次のとおりです (2014 年 1 月時点)。

表 1.1 バージョンと API レベル

バージョン	API Level	コード
Android 1.0	1	BASE
Android 1.1	2	BASE_1_1
Android 1.5	3	CUPCAKE
Android 1.6	4	DONUT
Android 2.0	5	ECLAIR
Android 2.0.1	6	ECLAIR_0_1
Android 2.1.x	7	ECLAIR_MR1
Android 2.2.x	8	FROYO
Android 2.3、2.3.1、2.3.2	9	GINGERBREAD
Android 2.3.3、2.3.4	10	GINGERBREAD_MR1
Android 3.0.x	11	HONEYCOMB
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.2	13	HONEYCOMB_MR2
Android 4.0、4.0.1、4.0.2	14	ICE_CREAM_SANDWICH
Android 4.0.3、4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.1、4.1.1	16	JELLY_BEAN
Android 4.2	17	JELLY_BEAN_MR1
Android 4.3	18	JELLY_BEAN_MR2
Android 4.4	19	KITKAT

第2章

Android のアーキテクチャとコンポーネント

この章の目的

- Android のアーキテクチャを理解する
- アプリケーションコンポーネントと Intent の概要を理解する
- アプリケーションコンポーネントのライフサイクルを理解する

2.1 Android のアーキテクチャ

Android は、次に示すように複数の階層から構成されます。

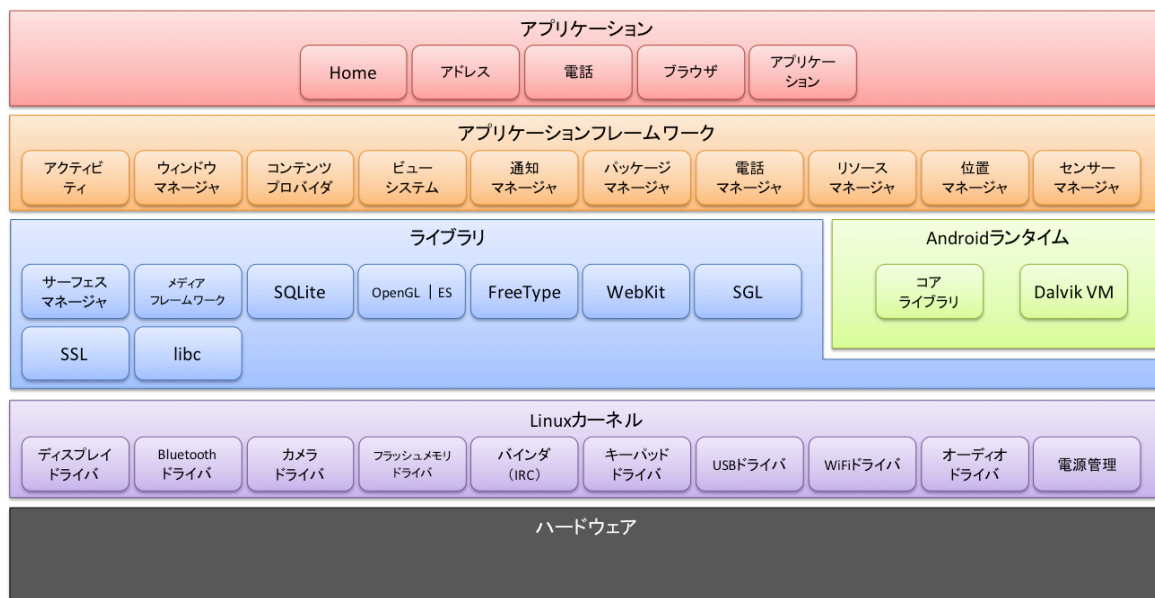


図 2.1 Android のアーキテクチャ

アプリケーション

メール、カレンダー、地図、ブラウザ、連絡先などのコアアプリケーション

Android のアーキテクチャ図で最も上のレイヤに配置されるのが、アプリケーションレイヤです。ユーザから見えるのはこれらのアプリケーションだけになっていることで内部で行われている動作のことは気づかずに利用することが出来ています。しかしながら、Android アプリケーション開発を行うにはその下のレイヤでどのようなことを行っているかを知らなければなりません。

アプリケーションフレームワーク

アプリケーション開発に利用可能な API

このレイヤは、アプリケーション開発のために使える高水準の部品を提供しています。必要に応じて独自コンポーネントを追加して拡張することが可能です。

アクティビティマネージャ (ActivityManager)

アプリケーションのライフサイクル (後述) を管理するものです。

コンテンツプロバイダ (ContentProvider)

アドレス帳のようにアプリケーションの間で共有しなければならないデータを参照できるようにしています。

リソースマネージャ (ResourceManager)

プログラム以外のデータをリソースといいます。「画像データ」「画面のレイアウト、ボタン、テキストボックスなどの設定ファイル」などを一元管理してリソース ID を指定してプログラムに引き渡すことが出来ます。

位置マネージャ (LocationManager)

位置情報をアプリケーションに提供してくれるもので、GPS を利用するものや基地局からの電波強度を利用するものなどがあります。

通知マネージャ (NotificationManager)

Notification とは、Android の画面上部に配置されているステータスバー上に表示することができるメッセージのことを指します。テキストメッセージだけでなく以下の設定をすることが出来ます。

- テキストメッセージ
- アイコン
- バイブレーション
- LED

ライブラリ

C/C++ のライブラリ

カーネルのすぐ上にあるこのレイヤは、Android ネイティブライブラリとして C か C++ で書かれている共有ライブラリです。

サーフェスマネージャ (SurfaceManager)

Android は単純なウィンドウマネージャを使用しています。表示サブシステムへのアクセスを管理して、複数のアプリケーションからシームレスに 2D と 3D グラフィックレイヤーを合成します。

2D/3D グラフィックスライブラリ (OpenGL ES)

OpenGL ES の API に準拠しています。ライブラリはハードウェア 3D アクセラレーション (ハードウェア依存)、あるいは標準装備の高度に最適化された 3D ソフトウェアラスライザーを使用します。

メディアフレームワーク (MediaFramework)

様々なフォーマットのビデオ録画・再生とオーディオ録音・再生をサポートしています。

SQLite

強力で軽量な様々なリレーショナルデータベースエンジンです。SQLite はデータ保存に単一ファイルを使用し、アプリケーションに組み込まれて使われます。データベースの操作は、プロトコルやプロセス間通信を使わずに API を呼び出すので高速に動作することができます。

Android ランタイム :

コアライブラリと Dalvik 仮想マシン

アプリケーションの実行環境は、Android ランタイムで実現されています。

コアライブラリ

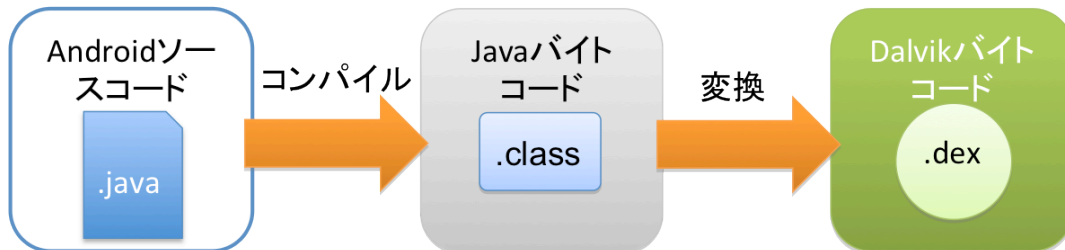
基本的な API を提供しています。Java SE 5.0 ライブラリを広域でサポートしていますが、グラフィカルなコンポーネントを使用したアプリケーションを開発する際に使用する swing 等の機能は取り除かれています。他に Android 専用ライブラリ、各種サードパーティのライブラリ (org.apache.http、org.xml.sax 等) を含んでいます。

Dalvik 仮想マシン

Java 言語で書かれた Android 用アプリケーションを動作させる仮想マシンです。メモリ容量が小さい携帯端末環境に最適化されています。実際には、Java 言語をコンパイルしてバイトコードしたものを実行します。使用する中間言語は Android 専用のものとなり、Dalvik Executable(DEX) と呼ばれています。つまり、一般的な Java バイトコードは利用していないため、Dalvik は Java 仮想マシン (JVM) ではありません。また、JVM は命令処理に使用するレジスタがスタック状になっ

ているのに対し、Dalvik は一般的な CPU のように任意のレジスタを選択する「レジスタ・ベース」となっています。

Android アプリケーションは主に Java で開発しますが、ソースコード (.java) をコンパイルして作成された Java バイトコード = クラスファイル (.class) を Dalvik バイトコード (.dex) に変換して Dalvik 仮想マシン上で実行します。



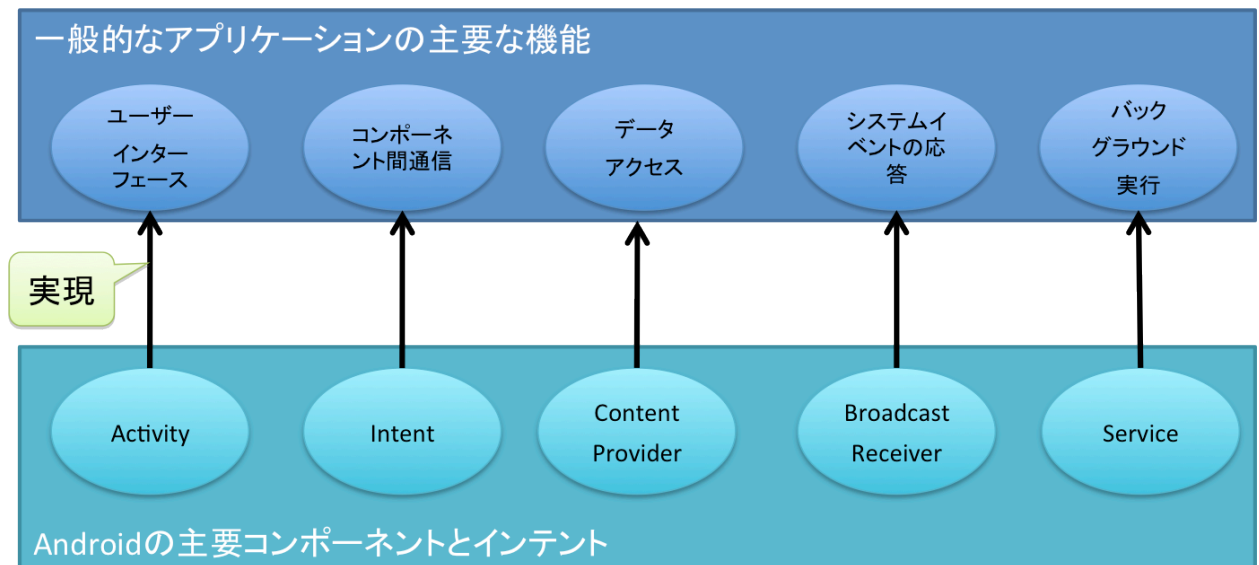
Linux カーネル

メモリ管理、プロセス管理、ネットワーク、ドライバなどのコアシステムサービス

2.2 アプリケーションコンポーネント

アプリケーションコンポーネントとは

Android フレームワークには主要な機能を実現させるためのアプリケーションコンポーネントと Intent が用意されています。アプリケーションコンポーネントとは、Android アプリケーションを開発するための部品です。用途と機能に応じてコンポーネントを使い分け、あるいは組み合わせることで、Android アプリケーションを開発します。アプリケーションコンポーネントには、Activity、Service、BroadcastReceiver、ContentProvider の 4 つがあります。これらのコンポーネントの起動や連携する仕組みとして Intent という機能が提供されています。



Activity

UI 画面を持つコンポーネントです。UI 画面を持つ Android アプリケーションは 1 つ以上の Activity から構成され、そのうちの 1 つがアプリケーションのエントリーポイントとして最初に呼び出されます。Java アプリケーションで main メソッドを持つクラスと同じです。

Activity は、`android.app.Activity` クラスを継承するサブクラスを定義して作成します。AndroidManifest.xml では `<activity>` タグで設定します。

Service

バックグラウンド処理を実行するコンポーネントです。たとえば、ファイルのダウンロード処理などを Activity で実行すると、処理が完了するまで UI 画面を操作できない状態になります。時間のかかる処理は Service に実装してバックグラウンドで実行するようにします。

Service は、`android.app.Service` クラスを継承するサブクラスを定義して作成します。AndroidManifest.xml では `<service>` タグで設定します。

BroadcastReceiver

システムで発生したイベントなどの非同期メッセージを受信するコンポーネントです。たとえば、バッテリー残量の低下などのイベントを示すメッセージを受け取り、それをユーザに通知するなどの目的で利用します。

BroadcastReceiver は、`android.content.BroadcastReceiver` クラスを継承するサブクラスを定義して作成します。AndroidManifest.xml では `<receiver>` タグで設定します。

ContentProvider

アプリケーション間でデータ共有を行うためのコンポーネントです。Android では、アプリケーションで作成したデータを異なるアプリケーションからアクセスすることは基本的に禁止されています。そのため、ContentProvider を作成して、自アプリケーションのデータを他のアプリケーションで共有できるようにします。また、連絡先やシステムの設定などのデータにアクセスするときには、Android が標準で提供している ContentProvider を利用します。

ContentProvider は、`android.content.ContentProvider` クラスを継承するサブクラスを定義して作成します。AndroidManifest.xml では<provider>タグで設定します。

Intent

アプリケーションコンポーネントのうち、Activity、Service、BroadcastReceiver は、Intent を使って他のアプリケーションコンポーネントから起動することができます。Intent とは、非同期メッセージを表すオブジェクトのことです。

Intent を利用する具体例

- アクティビティ A からアクティビティ B を起動する（画面遷移）
- アドレス帳アプリケーションから電話をかけるアプリケーションへ電話発信を依頼する

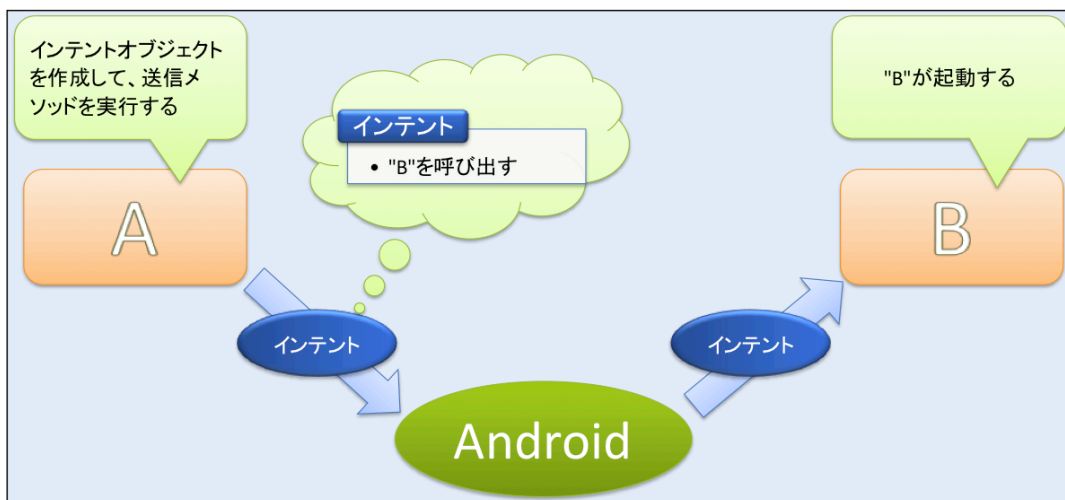


図: コンポーネント A からコンポーネント B を呼び出す例

明示的 Intent と暗黙的 Intent

呼び出し先のコンポーネントが明確な場合、コンポーネントを直接指定して呼び出す方法を「明示的 Intent」といいます。

一方、起動先を指定せず、実行する動作や発生したイベントを示すアクション（および関連するカ

カテゴリやデータ)のみを指定して生成した Intent を暗黙的 Intent といいます。暗黙的 Intent を利用する場合には、コンポーネント側でどの種類の Intent を受け取るかを設定しておかなければなりません。AndroidManifest.xml の<activity>タグ、<service>タグ、<receiver>タグの配下に<intent-filter>タグを記述し、<action>タグ、<category>タグ、<data>タグで受け取る Intent を定義します。



図: 明示的 Intent の例

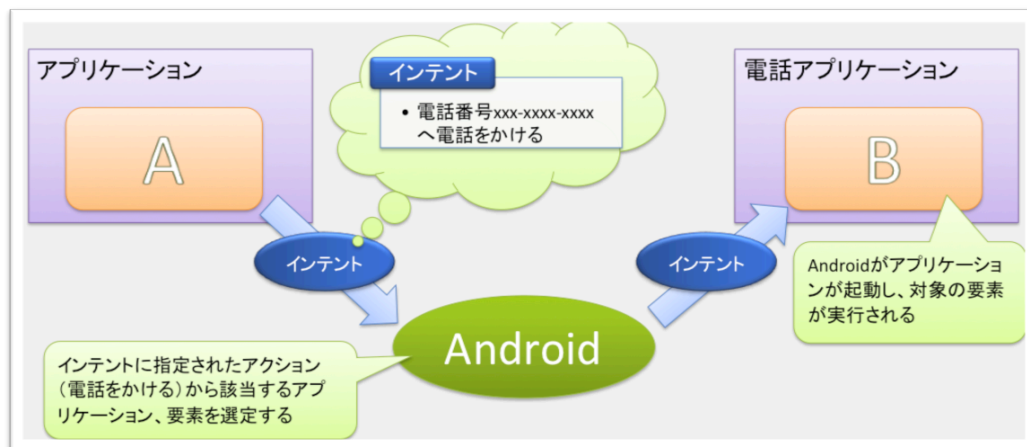


図: 暗黙的 Intent の例

暗黙的なIntentを使用すると、Androidに含まれているアプリケーションと連携して、次のような事が出来ます。

- 電話アプリケーション
 - 指定した番号のダイヤルウィンドウを開く
- アドレス帳アプリケーション
 - アドレス帳のリストを表示する
 - アドレス帳の中から1つのアドレス情報を表示する
 - 指定のアドレス情報を編集する

- Web ブラウザ
 - 指定した URL アドレスの Web サイトをブラウザで表示する

2.3 ライフサイクル

Activity、Service、BroadcastReceiver はライフサイクルを持ち、コンポーネントの起動や終了の際に特定のライフサイクルメソッドが呼び出されます。アプリケーションコンポーネントを開発するには、ライフサイクルメソッドがどのタイミングで呼び出されるかを理解する必要があります。

Activity のライフサイクル

Activity は起動されてから終了するまでの間、次の 3 種類の状態を遷移します。

フォアグラウンド状態:

アクティビティが表示されていて、ユーザとやりとりできる状態

バックグラウンド状態:

アクティビティが別のアクティビティによって隠されている状態

ビジブル状態:

アクティビティが表示されているが、操作できない状態

停止状態や一時停止状態の Activity は、メモリ不足の際にシステムによって強制的に終了される場合があります。

Activity では、これらの状態を遷移する際に特定のライフサイクルメソッドが呼び出されます。

#	イベント名	内容
1	onCreate	最初の起動時に発生するイベント
2	onStart	アクティビティが表示される直前に発生するイベント
3	onResume	アクティビティが利用可能な状態(アクティブ状態)になる直前に発生するイベント
4	onPause	アクティビティがビジブルになる直前に発生するイベント
5	onStop	onPauseの後、Activityが非表示の状態になった場合に発生するイベント (メモリ不足の際、イベントが発生しない場合がある)
6	onRestart	終了状態のアクティビティが再度表示される際に発生するイベント
7	onDestory	アクティビティが破棄される直前に発生するイベント (メモリ不足の際、イベントが発生しない場合がある)

図: 通知されるイベント

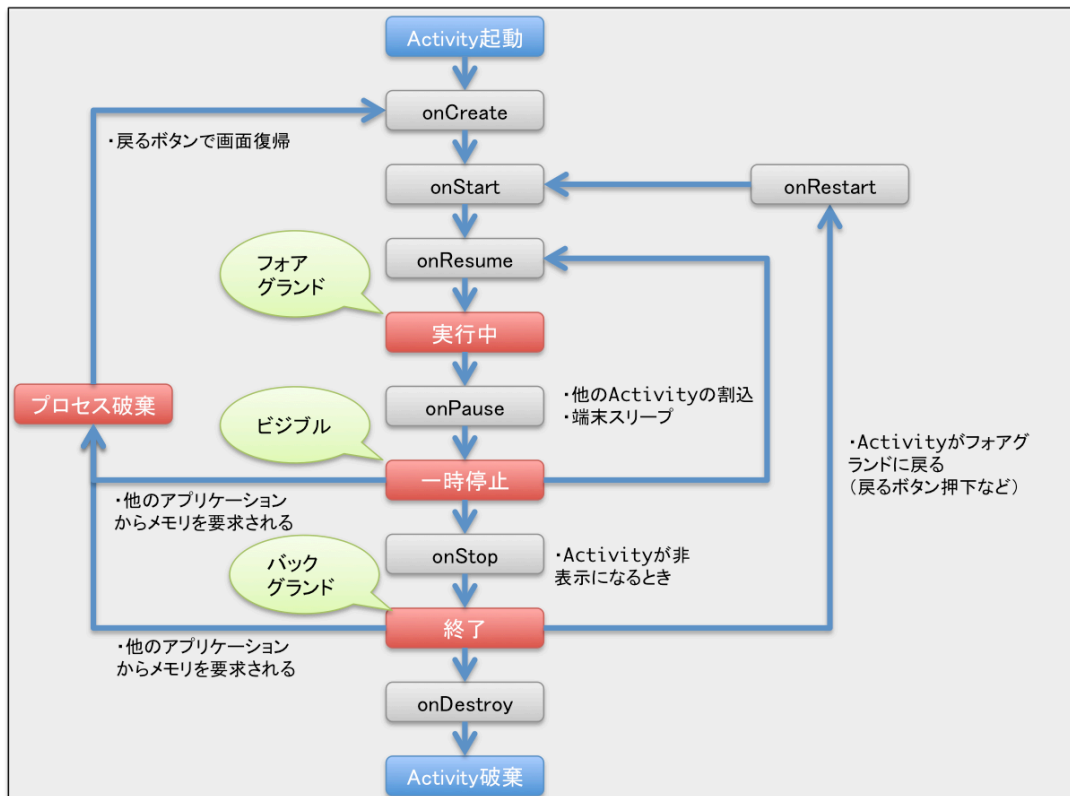


図: Activity のライフサイクル

Activity を作成する際には、必要に応じてこれらのライフサイクルメソッドをオーバーライドします。たとえば、Activity の実行に必要な初期処理は onCreate メソッドで実行します。また、Activity の終了前に実行しておかなければならない処理（データの保存など）は onPause メソッドに記述します。これは、別の Activity がフォアグラウンド状態になり、現在の Activity で onPause メソッドが呼び出された後は、メモリ不足などの理由によりシステムによって強制終了され、onStop メソッドと onDestroy メソッドが呼び出されない可能性があるからです。

Service のライフサイクル

Service のライフサイクルは、どのように起動されたかによって異なります。Service が startService メソッドにより起動されると、onCreate メソッド、onStartCommand メソッドが呼び出されます。stopService メソッドまたは stopSelf メソッドにより終了するときには onDestroy メソッドが呼び出されます。Service が bindService メソッドによりバインドされると、onCreate メソッド、onBind メソッドが呼び出されます。unbindService によりバインドが解除されると、onUnbind メソッド、onDestroy メソッドが呼び出されます。

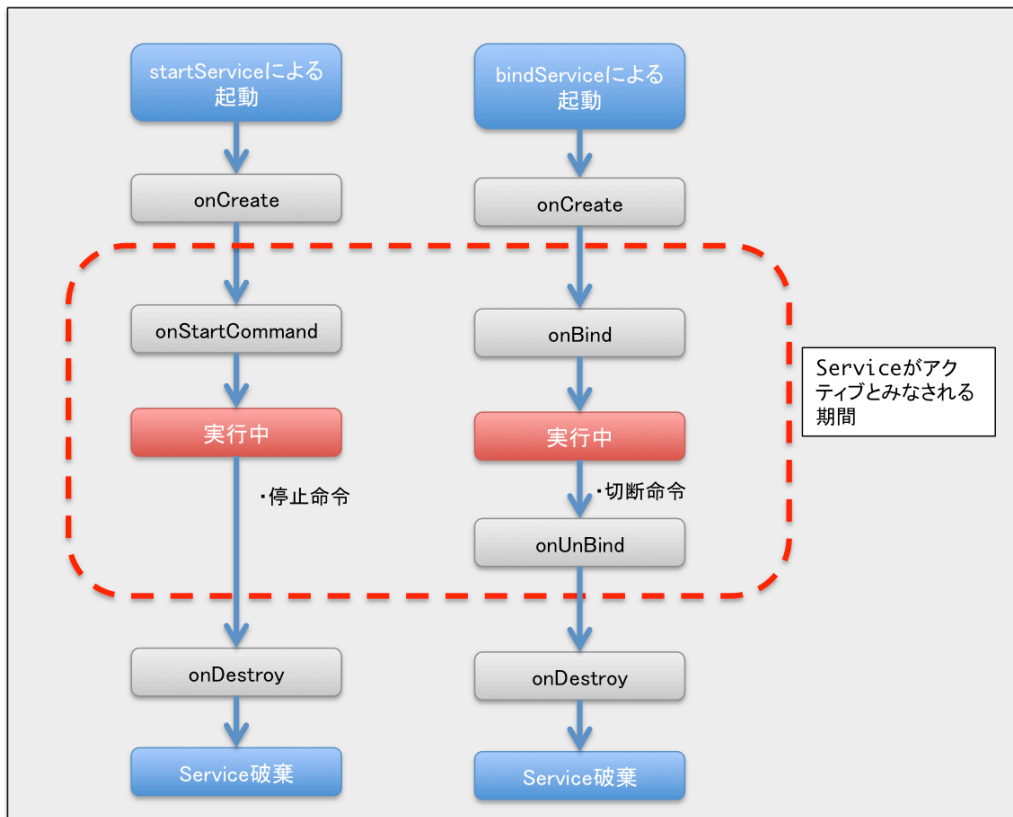


図: Service のライフサイクル

BroadcastReceiver のライフサイクル

BroadcastReceiver はライフサイクルメソッドとして onReceive メソッドのみを持ちます。BroadcastReceiver が起動されると onReceive メソッドが呼び出されます。onReceive メソッドが呼び出されてから終了するまでの期間のみ、BroadcastReceiver はアクティブとみなされます。

第3章

開発環境の構築

この章の目的

- Android アプリケーションの開発環境を準備する
- エミュレータを作成する
- 環境変数を登録する

3.1 Android アプリケーションの開発環境を準備する

3.2 開発環境とインストールツールについて

本書で使用する PC の開発環境とインストールツールについて説明します。

PC の開発環境

本書で使用している Android 開発環境は以下のとおりです。

この章では、Android SDK、SDK Platform をインストールします。

表 3.1 開発環境

OS	Windows 7
作業ディレクトリ	c:\¥android_training_basic
Java SDK	JDK 1.6
AndroidSDK	Ver 22.3
SDK Platform	Android 4.4 (API 19)

- <注意>
 - c:\¥android_training_basic フォルダは各自で作成すること
 - Java は事前にインストールされていること

付属 DVD について

演習に必要な全てのツールは、本書付属の DVD で提供済です。

DVD の中に「android_training_basic.zip」というファイルが1つ用意されています。

zip ファイルを解凍すると「android_trainig_basic」フォルダが作成され、その中には表 3.2 ようなファイル及びフォルダが用意されています。

表 3.2 android_trainig_basic.zip の中身

ディレクトリ名	説明
adt-bundle-windows-x86_64-20131030.zip	Android 開発ツールと Eclipse(64bit)
adt-bundle-windows-x86-20131030.zip	Android 開発ツールと Eclipse(32bit)
workspace	Eclipse のワークスペース
answer_docs\html	実習の解答ドキュメント

インストールするツール

本書では、表 3.3 のツールをインストールします。

表 3.3 開発環境

ソフトウェア	バージョン
Eclipse	Eclipse IDE with built-in ADT
AndroidSDK	Ver 22.3
SDK Pratform SDK	Android 4.4 (API 19)

開発ツールの説明

Eclipse IDE with built-in ADT:

Android アプリケーション開発に対応した Eclipse です。

Android の開発をするための Eclipse のプラグイン ADT が組み込まれています。

Android SDK:

Android 上で実行可能なアプリケーション開発するための SDK (Software Development Kit) です。

Android 端末やエミュレータとホスト PC を USB で接続して、アプリケーション・プログラムを携帯電話機上で実行しながら PC 上でデバッグすることが可能です。

Android Development Tools (ADT):

Eclipse で Android の開発をするための Eclipse のプラグインです。

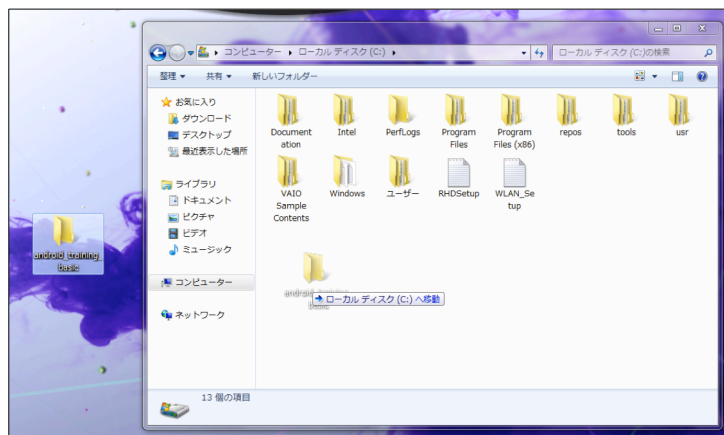
3.3 開発環境の構築

手順

1. AndroidSDK、Eclipse のインストール
2. Eclipse の設定
3. 環境変数の登録

手順 1. AndroidSDK、Eclipse のインストール

配布ファイル「android_training_basic.zip」を解凍し、作成された「android_training_basic」フォルダを C ドライブ直下に配置します。



c:¥android_training_basic 以下が、図 3.1 のようになっていることを確認します。

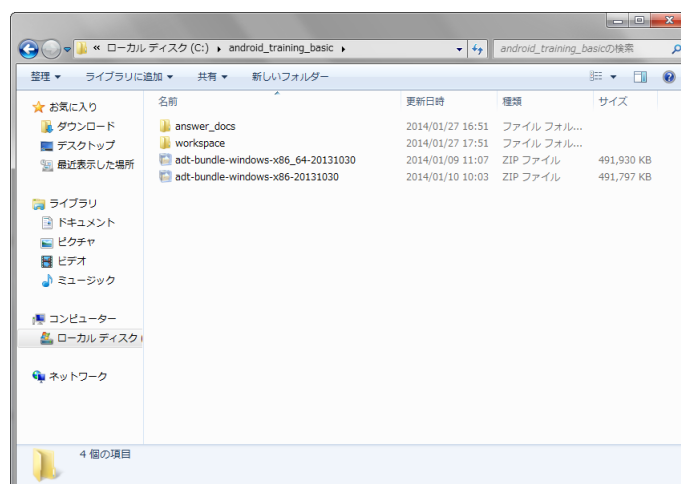


図 3.1 c:¥android_training_basic

次に、「adt-bundle-windows-x86_64-20131030.zip^{*1}」を同じフォルダ内に解凍します。作成された adt-bundle-windows-x86_64-20131030 フォルダの内容が、図 3.2 の用になっていることを確認します。「adt-bundle-windows-x86_64-20131030.zip」には、Android 開発に必要なツールが用意されています。それらのバージョン情報は、表 3.3 のようになっています。

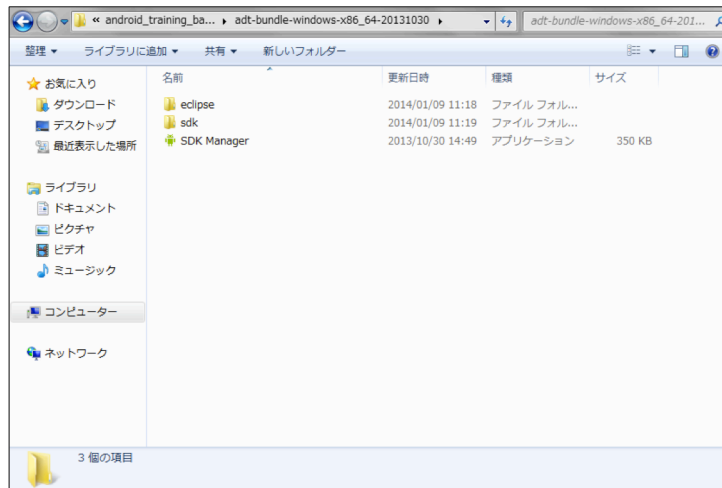
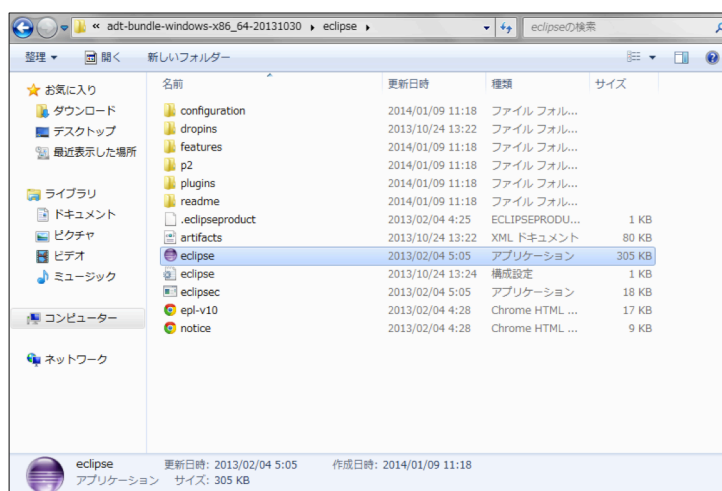


図 3.2 c:\¥android_training_basic¥adt-bundle-windows-x86_64-20131030 フォルダの内容

これでインストールは完了です。

手順 2. Eclipse の設定

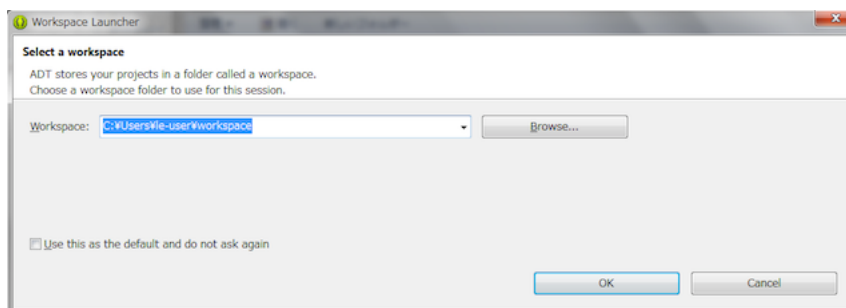
c:\¥android_training_basic¥adt-bundle-windows-x86_64-20131030¥eclipse 以下にある、eclipse をダブルクリックして起動します。



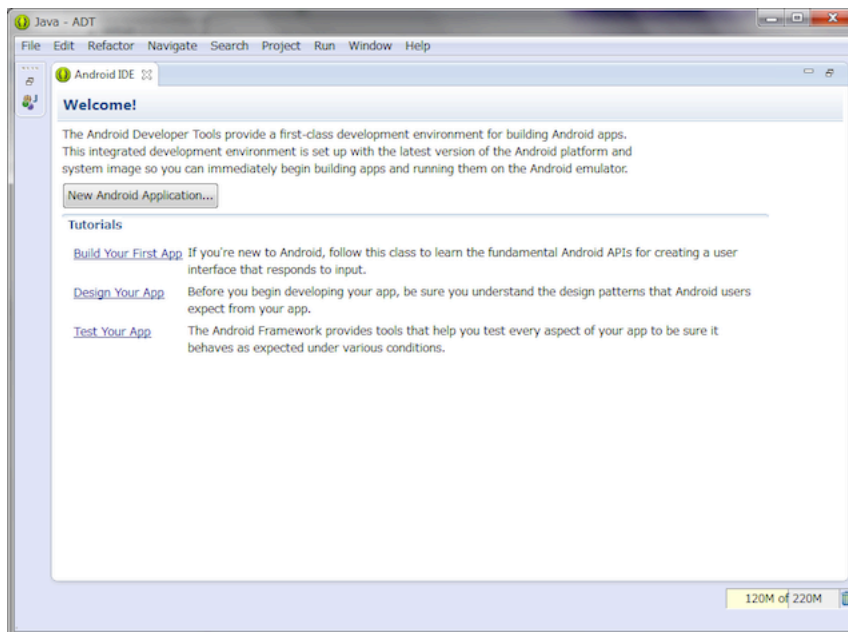
^{*1} 32bit 版の Windows を使っている場合は、adt-bundle-windows-x86-20131030.zip を使用してください。



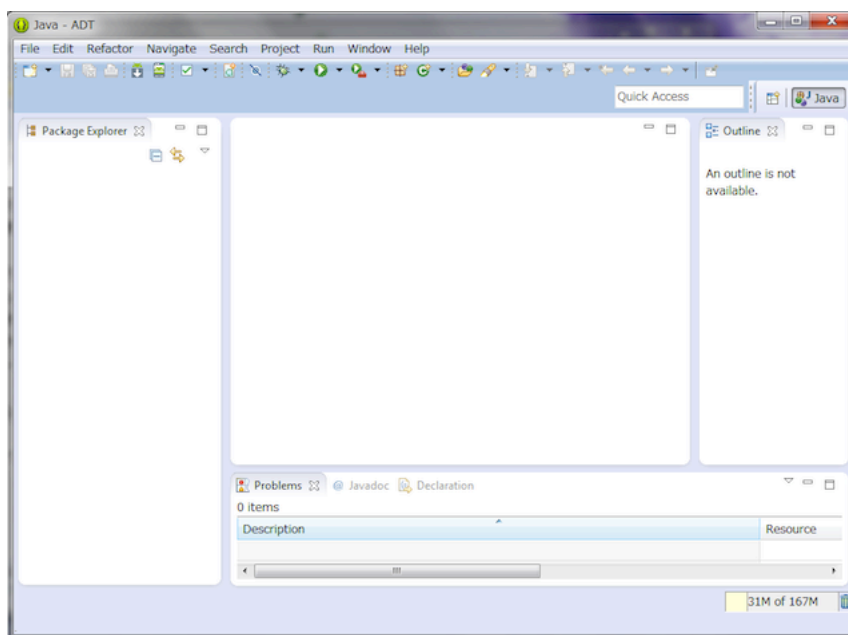
Workspace 選択ウィンドウが表示されたら、Workspace に
「C:¥android_training_basic¥workspace」
と入力し、 [Use this as the default and do not ask again] にチェックを入れて [OK]
ボタンをクリックします。



Welcome!と表示されたら、 [Android IDE ×] タブの "×" をクリックして閉じます。



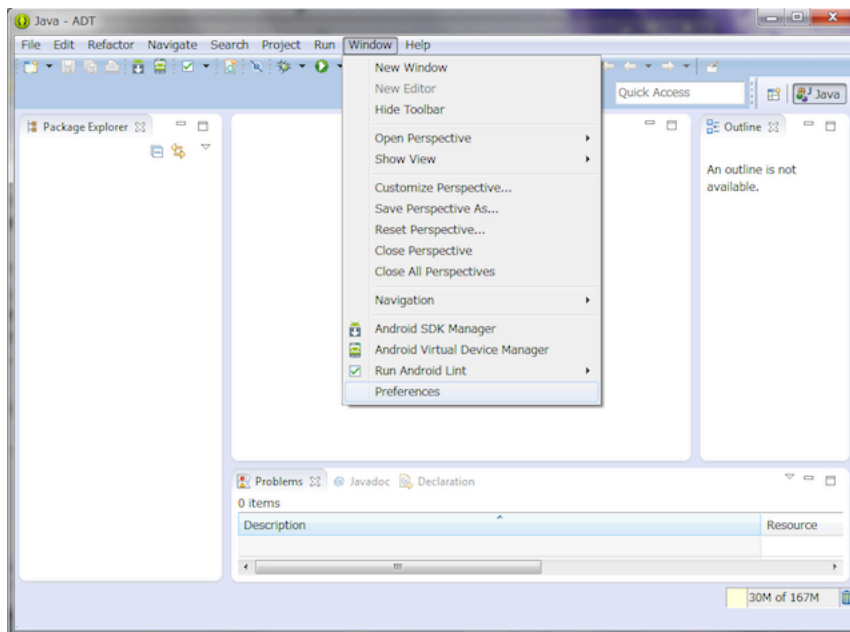
画面が次のように表示されていることを確認します。



ADT の設定

Eclipse の Android SDK のパスを確認します。

[Window] メニュー > [Preferences] を選択し、Preferences ウィンドウを表示します。

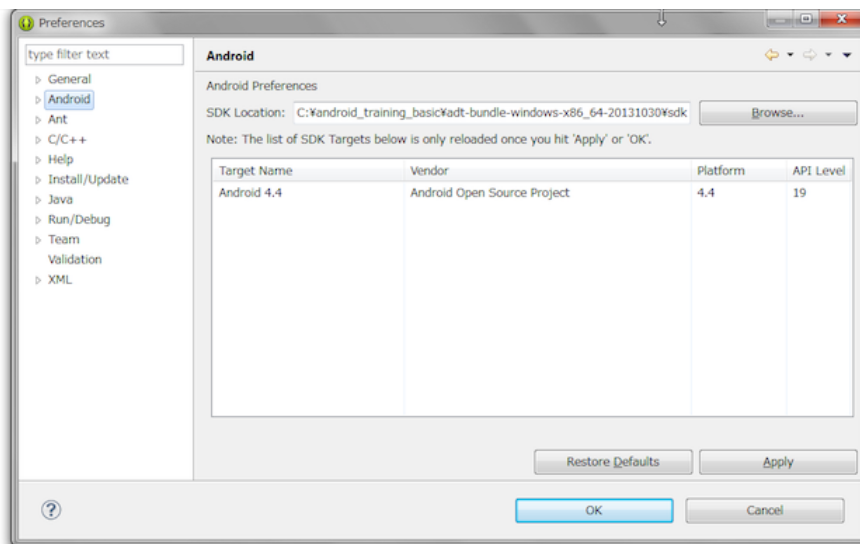


Preferences ウィンドウから [Android] を選択します。

SDK Location の内容が

c:\¥android_training_basic¥adt-bundle-windows-x86_64-20131030¥sdk

となっていることを確認します。

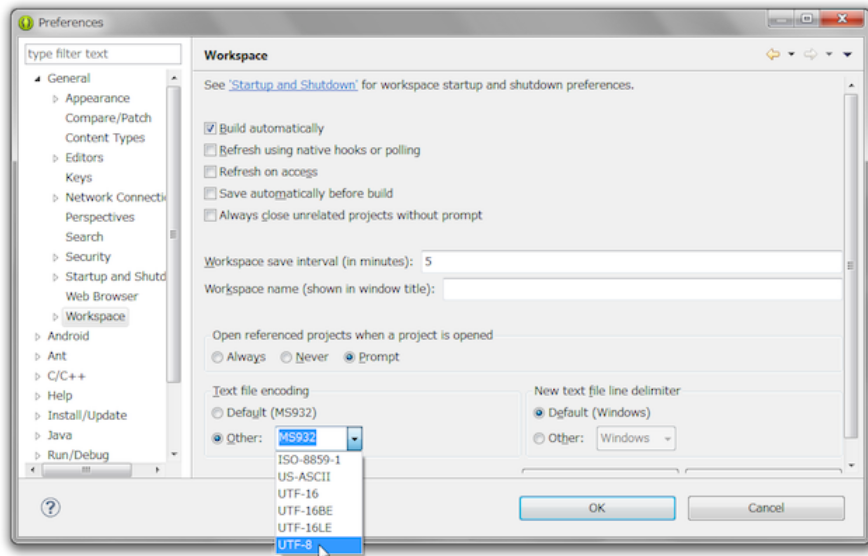


文字コードの設定

次に、文字コードの設定を変更します。

Preferences ウィンドウから [General] > [Workspace] を選択し [Text file encoding] の値を [UTF-

8] に変更します。



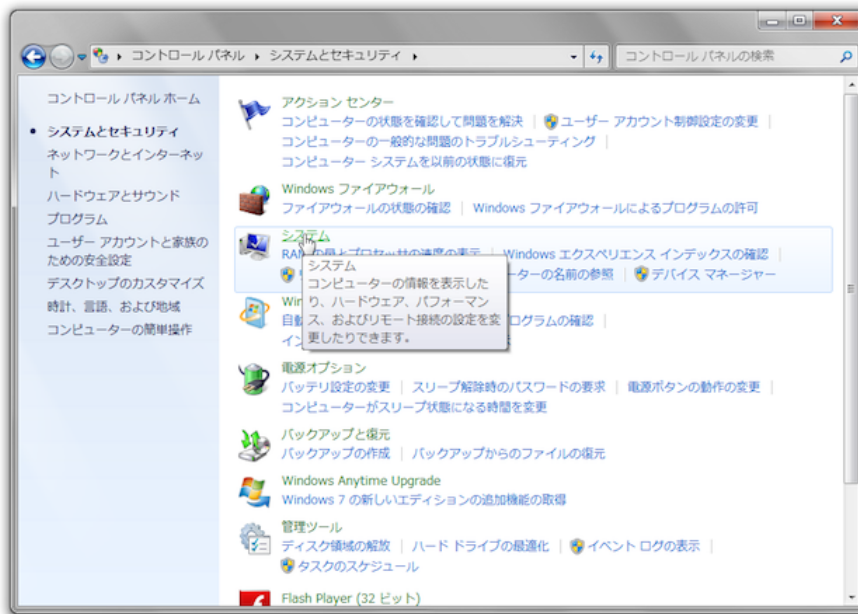
以上で Eclipse の設定は完了です。

手順 3. 環境変数の登録

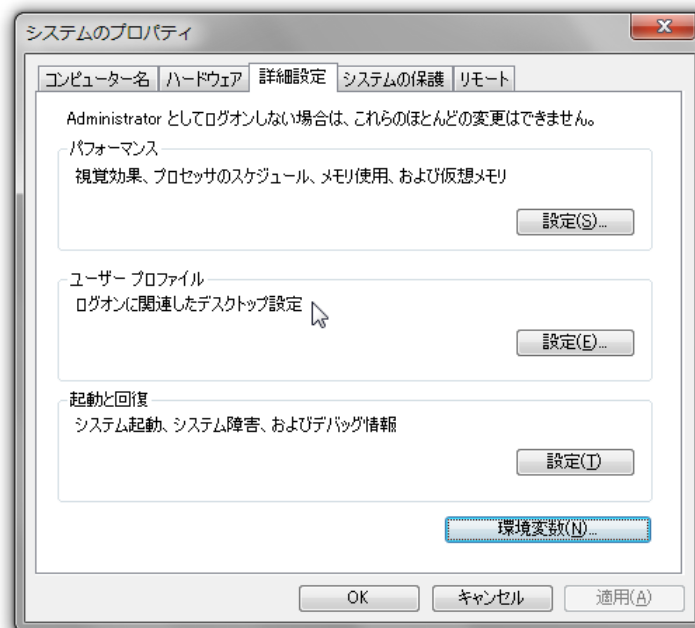
スタートメニューの [コントロールパネル] からコントロールパネルを開き、[システムとセキュリティ] を選択します。



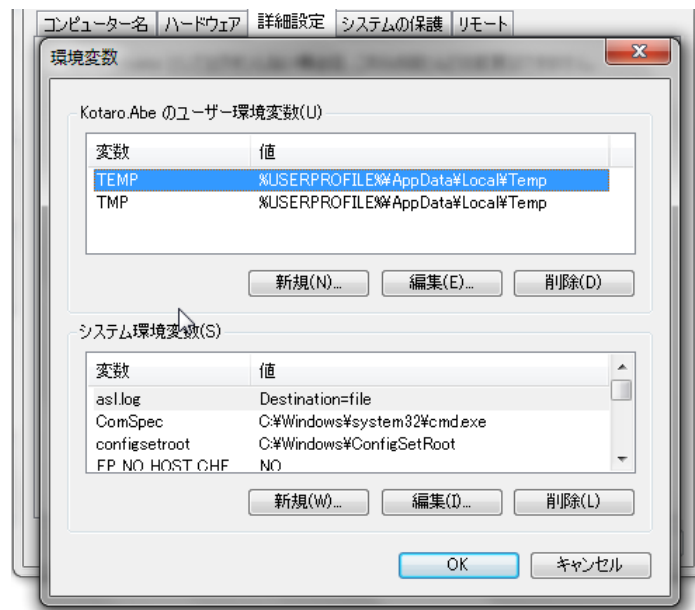
システムとセキュリティウィンドウで [システム] を選択します。



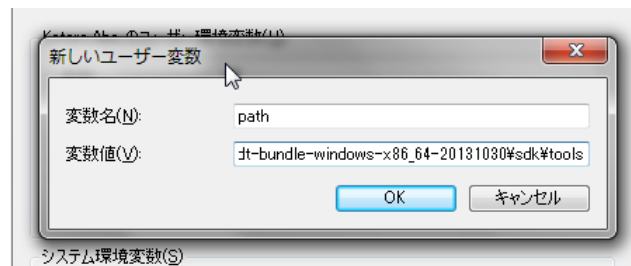
[システムの詳細設定] を選択し、システムのプロパティウィンドウを表示させます。



システムのプロパティウィンドウで [環境変数] ボタンをクリックし環境変数ウィンドウを表示させます。



環境変数ウィンドウの [ユーザー環境変数] から [新規] ボタンをクリックし、新しいユーザー変数ウィンドウを表示させます。



新しいユーザー変数ウィンドウの、変数名と変数値を次のように設定します。

- 変数名
 - path
- 変数値
 - C:¥ android_training_basic ¥ adt-bundle-windows-x86_64-20131030 ¥ sdk ¥ platform-tools;C:¥ android_training_basic ¥ adt-bundle-windows-x86_64-20131030 ¥ sdk ¥ tools

OK ボタンをクリックして全てのウィンドウを閉じます。

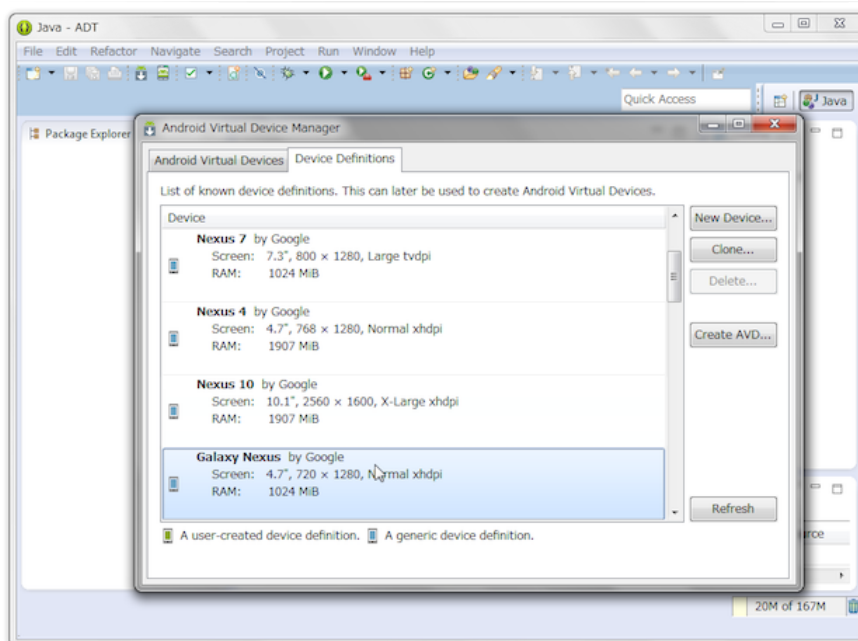
3.4 エミュレータの作成

Android アプリケーション開発では、Android 端末またはエミュレータで開発したアプリケーションのデバッグやテストを行います。Android エミュレータとは、パソコン上で実行可能な仮想端末ソフトウェアです。エミュレータを実行するには、AVD (Android Virtual Device) を作成します。

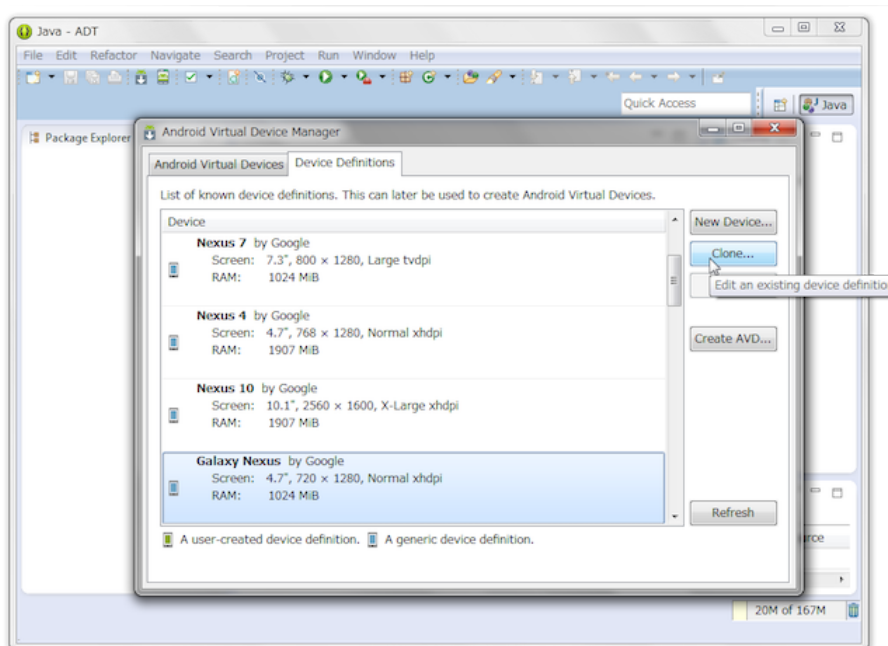
AVD の定義を作成する

ツールバーの [Android Virtual Device Manager] ボタンをクリックして Android Virtual Device Manager を起動します。

[Device Definitions] タブを選択し、定義一覧を表示します。



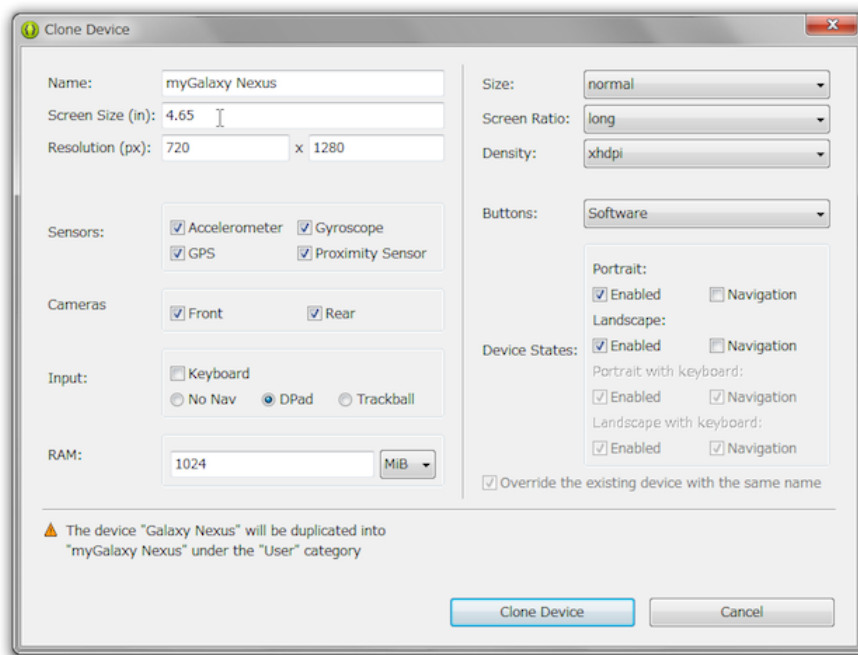
一覧から [Galaxy Nexus by Google] を選択し、[Clone] ボタンをクリックします。



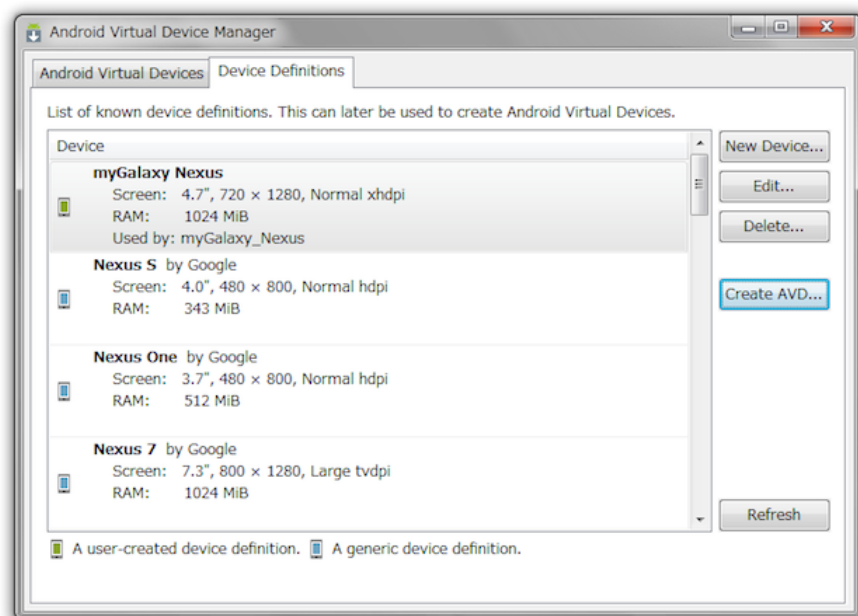
Clone Device ウィンドウで次の項目を変更し、[Clone Device] ボタンをクリックします。

表 3.4 定義の設定

項目	設定値
Name	myGalaxy Nexus
Input	DPad

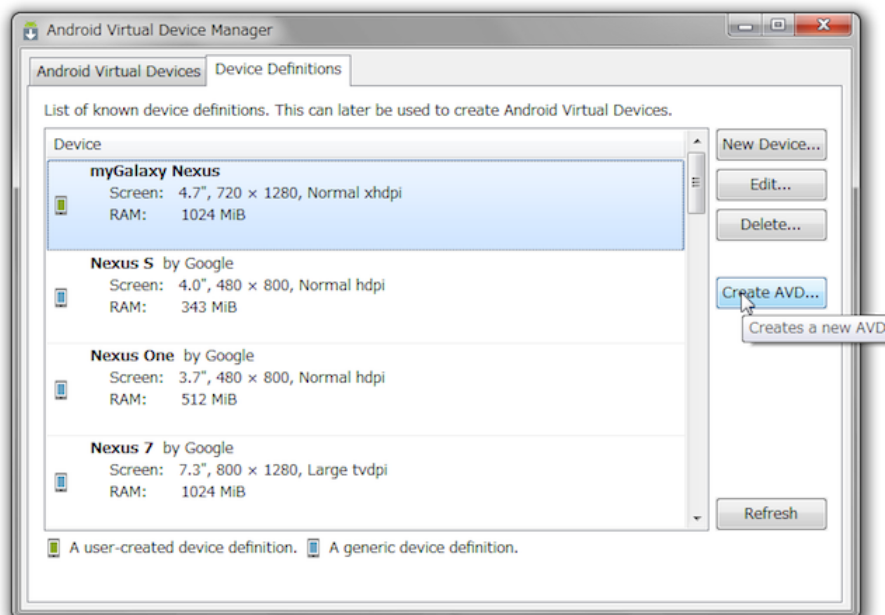


一覧に myGalaxy Nexus が追加されていることを確認します。



AVD を作成する

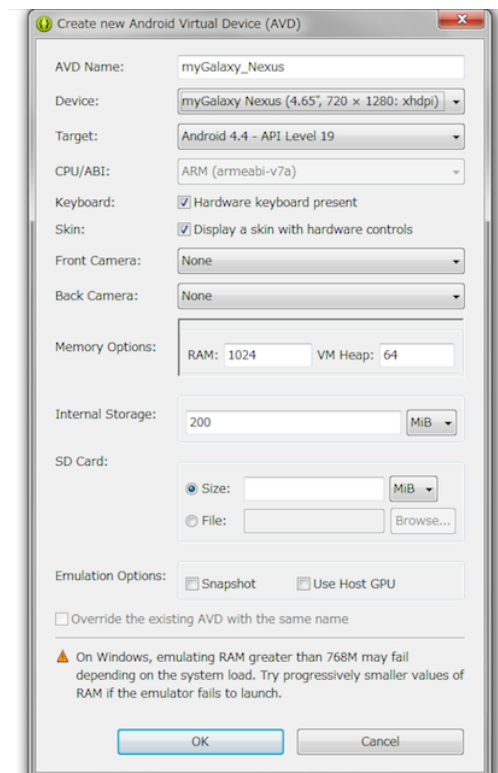
一覧から myGalaxy Nexus を選択して、[Create AVD...] ボタンをクリックします。



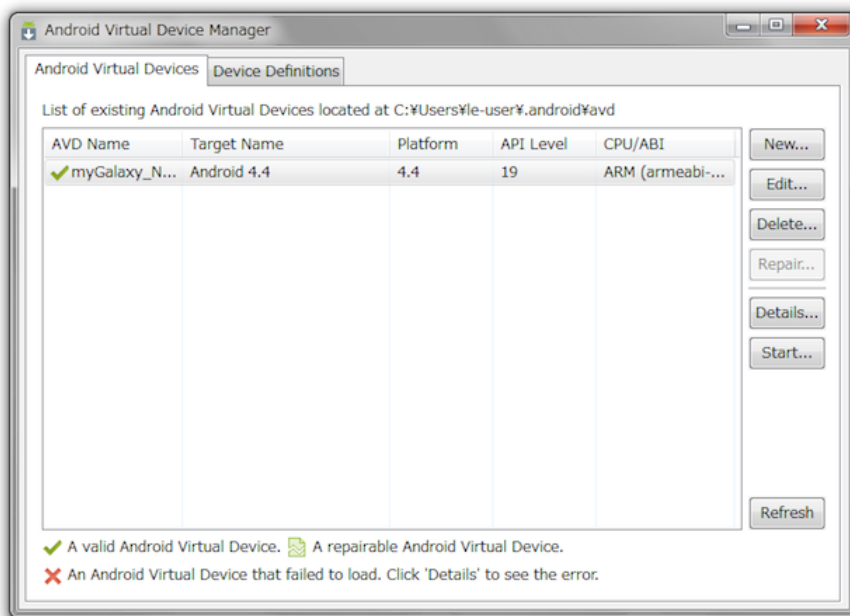
Create new Android Virtual Device ウィンドウで次のように設定し、[OK] ボタンをクリックします。

表 3.5 AVD の設定

項目	設定値
AVD Name	myGalaxy_Nexus
Device	myGalaxy Nexus
Target	Android 4.4

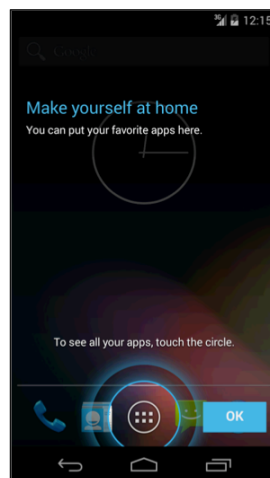
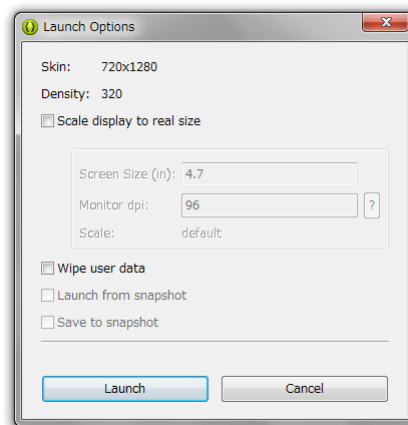


[Android Virtual Devices] タブに切り替え、一覧に myGalaxy_Nexus が追加されていることを確認します。



AVD を起動する

作成した myGalaxy_Nexus を選択して、[Start...] ボタンをクリックします。
Launch Options ウィンドウで [Launch] ボタンをクリックすると、エミュレータが起動します。



第 4 章

開発ツールの使い方

目的

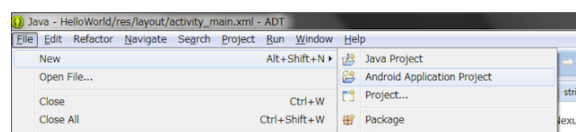
- Eclipse の基本的な操作方法をおぼえる
- 新規プロジェクトを作成できる
- 画面デザインを変更することができる
- ログの出力ができるようになる

4.1 アプリケーションの作成

Eclipse を使って簡単な Android アプリケーションを作成してみましょう。Eclipse で Android アプリケーションを作成するには Android プロジェクトを作成します。

プロジェクトの新規作成

Eclipse を起動し、[File]メニュー > [New] > [Android Application Project]を選択します。

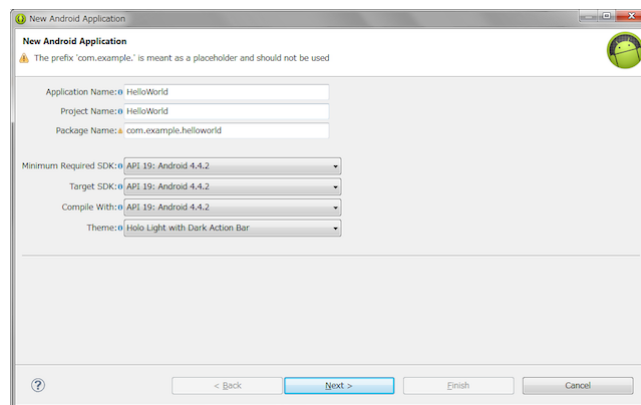


新規作成したプロジェクトは、図 4.1 のような Hello world! と表示されるアプリケーションとして作成されます。

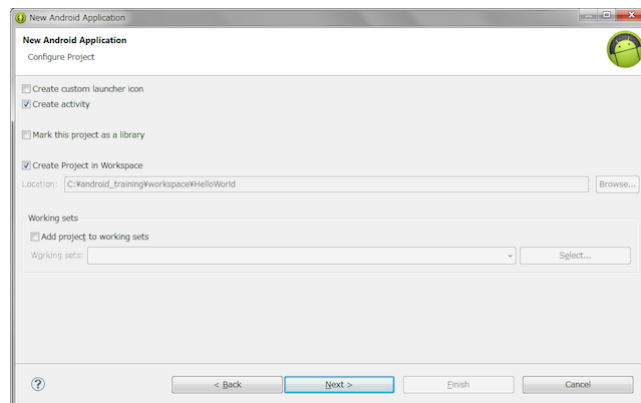
[New Android Application] ウィンドウでは、Application の設定情報を表 4.1 次のようにし、[Next] ボタンをクリックします。

表 4.1 プロジェクト概要

項目	設定値
Application Name	HelloWorld
Project Name	HelloWorld
Package Name	com.example.helloworld
Minimum Required SDK	API 19: Android 4.4.2
Target SDK	API 19: Android 4.4.2
Compile With	API 19: Android 4.4.2
Theme	Holo Light with Dark Action Bar



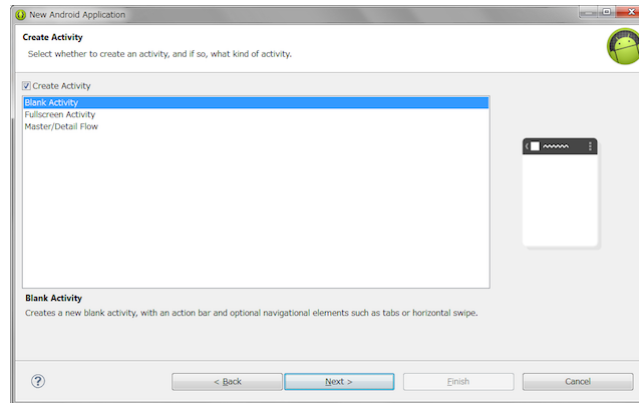
次の画面では、[Create Custom launcher icon] のチェックを外し、[Next] ボタンをクリックします。



[Create Activity] ウィンドウでは、デフォルトで表 4.2 のような設定になっていることを確認し、[Next] ボタンをクリックします。

表 4.2 Create Activity の設定

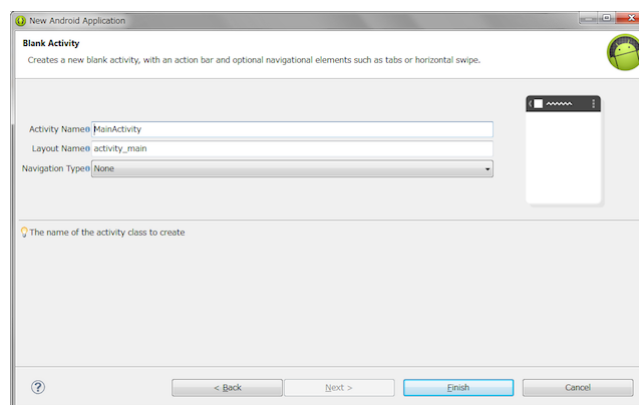
項目	設定値
Create Activity	チェック有り
Activityの種類	Blank Activity



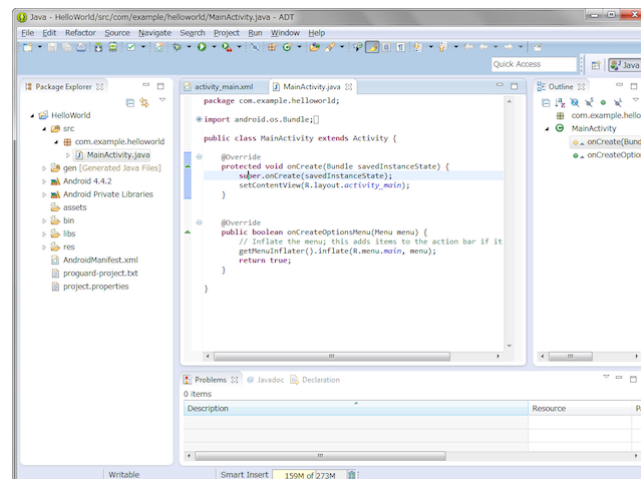
[New Blank Activity] ウィンドウでは、デフォルトで表 4.3 のような設定になっていることを確認し、[Finish] ボタンをクリックする

表 4.3 New Blank Activity の設定

項目	設定値
Activity Name	MainActivity
Layout Name	activity_main
Navigation Type	None



新規プロジェクトが作成され、[Package Explorer] に Hello World プロジェクトが追加されていることを確認します。



Android プロジェクトの構造

Android プロジェクトを作成すると、プロジェクト内にさまざまなフォルダやファイルが自動的に作成されます。[Package Explorer] で HelloWorld プロジェクトを展開してみましょう。

src フォルダ:

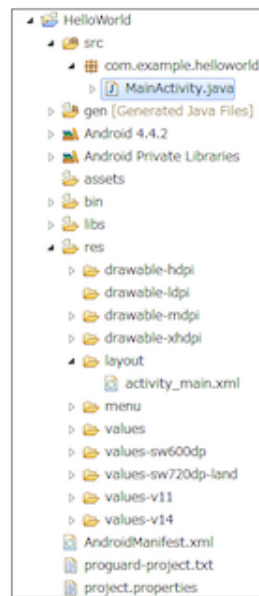
Android アプリケーションを構成する Activity のソースコード (MainActivity.java) が生成されます。

res フォルダ:

Android アプリケーションで利用するリソースを格納するフォルダです。レイアウト、メニュー、文字列などのリソースが自動的に生成されます。

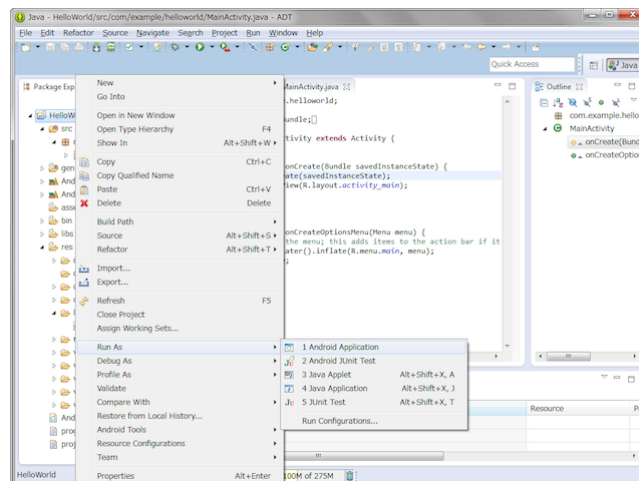
AndroidManifest.xml:

Android アプリケーションの設定を行うためのファイルです。



アプリケーションの実行

作成された、HelloWorld プロジェクトを選択し、右クリックし、 [Run As] > [Android Application] を選択します。



アプリケーションがエミュレータにインストールされ、Hello World アプリケーションが起動するのを確認します。



図 4.1 HelloWorld

4.2 Activity

プロジェクトの新規作成で、[Create Activity] にチェックを入れて、Android プロジェクトを作成すると、自動的に Activity のソースコードが生成されます。Activity は、Android アプリケーションを構成するコンポーネントの 1 つで、UI 画面を持ちます。Android アプリケーションは基本的に 1 つ以上の Activity で構成します。

HelloWorld プロジェクトでは MainActivity が自動的に生成されました。アプリケーションを実行すると、この MainActivity が最初に起動されます。

MainActivity クラスのコードは次のようになっています。

リスト 4.1: MainActivity.java

```
1: package com.example.helloworld;
2:
3: import android.os.Bundle;
4: import android.app.Activity;
5: import android.view.Menu;
6:
7: public class MainActivity extends Activity {
8:
9:     @Override
10:    protected void onCreate(Bundle savedInstanceState) {
11:        super.onCreate(savedInstanceState);
12:        setContentView(R.layout.activity_main);
13:    }
14:
15:    @Override
16:    public boolean onCreateOptionsMenu(Menu menu) {
17:        // Inflate the menu; this adds items to the action bar if it is present.
18:        getMenuInflater().inflate(R.menu.main, menu);
19:        return true;

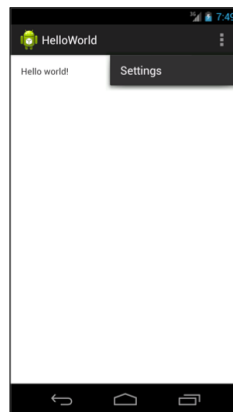
```

```
20:     }  
21:  
22: }
```

10行目の `onCreate` メソッドは `Activity` のライフサイクルメソッドの1つです。`Activity` が起動されると、最初に `onCreate` メソッドが呼び出されます。

`onCreate` メソッドでは、Android アプリケーションの実行に必要な処理を行います。12行目の `setContentView` メソッドは、レイアウトを設定するメソッドです。`Activity` が表示する画面レイアウトは、このメソッドに指定するレイアウトファイル（詳細は後述）に指定します。

16行目の `onOptionsItemSelected` メソッドは、アプリケーションでオプションメニューを利用する際に必要なメソッドです。オプションメニューとは、[MENU] ボタンを押したときに表示されるメニューのことです。HelloAndroid プロジェクトでは次のようなメニューが表示されます。



4.3 リソースファイル

Android プロジェクトを作成すると、プロジェクト内に `res` フォルダが作成されます。`res` フォルダは、リソースファイルを格納するフォルダです。

Android アプリケーションでは、画面レイアウト、文字列、メニューなどを XML で定義してリソースとして利用します。これらのリソースは、プログラムや他のリソースファイルからリソース ID で参照できます。

レイアウトファイル

画面レイアウトを定義するファイルです。HelloAndroid プロジェクトでは `res/layout` フォルダに `activity_main.xml` というレイアウトファイルが生成されます。

リスト 4.2: `activity_main.xml`

表 4.4 リソースファイルの種類

リソースファイル	内容
res/drawable(-mdpi、-hdpi など)/*.*	画像ファイル (.png、.jpg など)
res/layout/*.xml	画面デザイン情報 (activity_main.xml など)
res/values/colors.xml	色情報
res/values/dimens.xml	サイズ情報
res/values/strings.xml	文字列情報

```

1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent"
5:     android:paddingBottom="@dimen/activity_vertical_margin"
6:     android:paddingLeft="@dimen/activity_horizontal_margin"
7:     android:paddingRight="@dimen/activity_horizontal_margin"
8:     android:paddingTop="@dimen/activity_vertical_margin"
9:     tools:context=".MainActivity" >
10:
11:     <TextView
12:         android:layout_width="wrap_content"
13:         android:layout_height="wrap_content"
14:         android:text="@string/hello_world" />
15:
16: </RelativeLayout>

```

4.4 画面デザインを変更する

ADT にはレイアウトファイルを編集するためのレイアウトエディタが提供されています。レイアウトエディタの機能を使用して、HelloWorld にボタン、チェックボックス、エディットテキスト (テキスト入力ができるビュー) を含む画面を作成してみましょう。

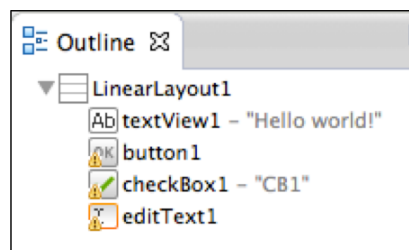
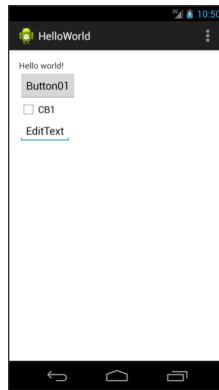


図 4.2 アウトライン

表 4.5 各ビューの表示文字列

ビュー	表示文字列
ボタン	Button01
チェックボックス	CB1
エディットテキスト	EditText



画面デザインリソースファイルを開く

Package Explorer から `res/layout/activity_main.xml` をダブルクリックします。
`res/layout/activity_main.xml` のファイルがレイアウトエディタで開かれ、図 4.3 の画面が表示されることを確認します。

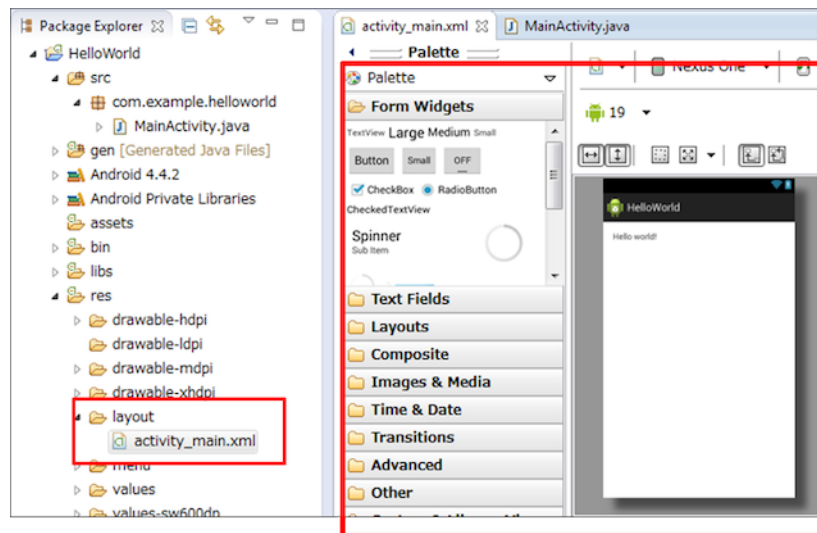
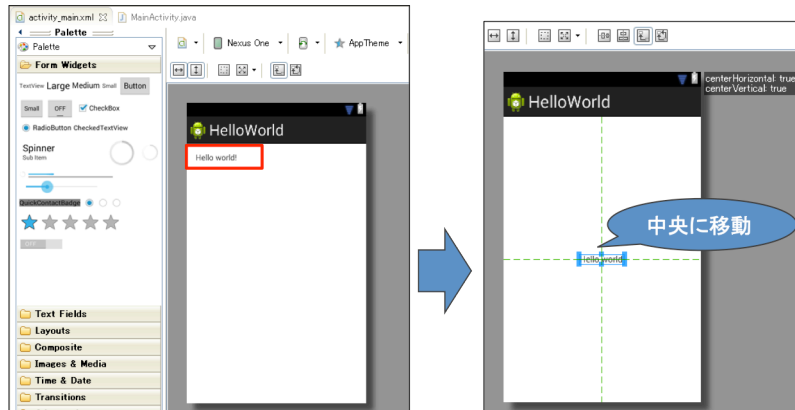


図 4.3 レイアウトエディタ

TextView を画面中央に移動する

レイアウトエディタの Graphical Layout タブで、TextView を画面中央に移動させます。

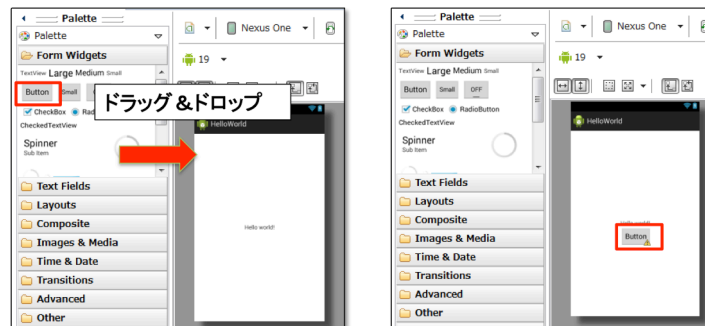
- プレビュー画面上の TextView を選択し、中央にドラッグします。



Button、CheckBox を追加する

画面デザインにボタン、チェックボックスを追加します。

- [Form Widgets] から Button を選択し、スクリーン上にドラッグ & ドロップし、画面に Button が表示されていることを確認する
- [Form Widgets] から CheckBox をスクリーン上にドラッグ & ドロップし、画面に CheckBox が表示されていることを確認する



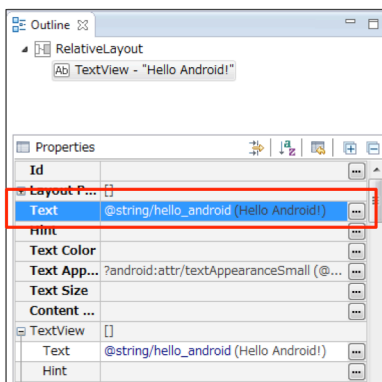
Button、CheckBox のテキストを変更する

次に、配置した Button と CheckBox のテキストを変更します。

- スクリーン上の CheckBox を選択し、Properties ビュー上で Text プロパティを CheckBox

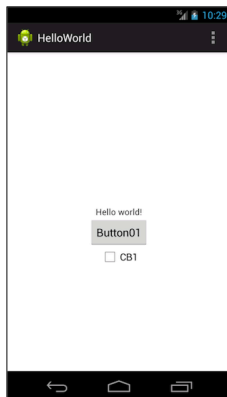
から CB1 に変更する

- 同様に Button の Text プロパティを Button01 に変更する



アプリケーションを実行する

変更を保存し、アプリケーションを実行します。実行画面が次のようになっていることを確認します



4.5 View の整列

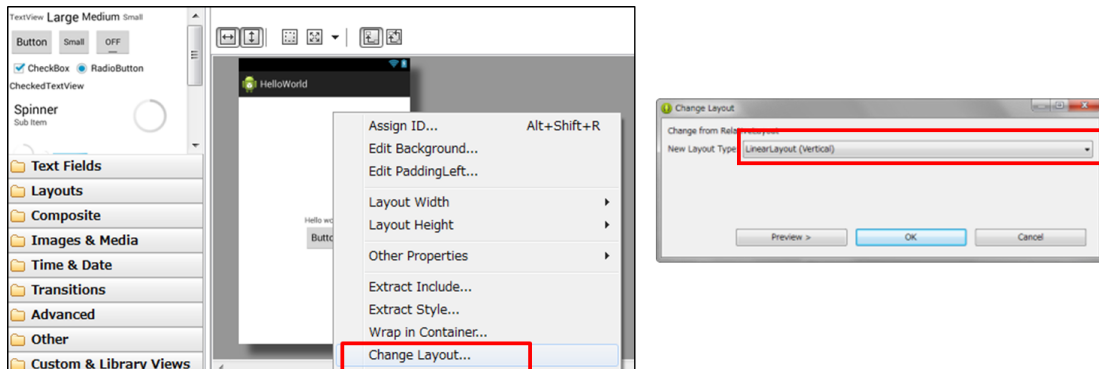
Layout を変更し、ボタンやテキストを整列させましょう。

LinearLayout に変更する

RelativeLayout から LinearLayout に変更します。

- プレビュー画面上で右クリックし、 [Change Layout] を選択

- New Layout Type を LinearLayout(Vertical) に変更



アプリケーションの実行

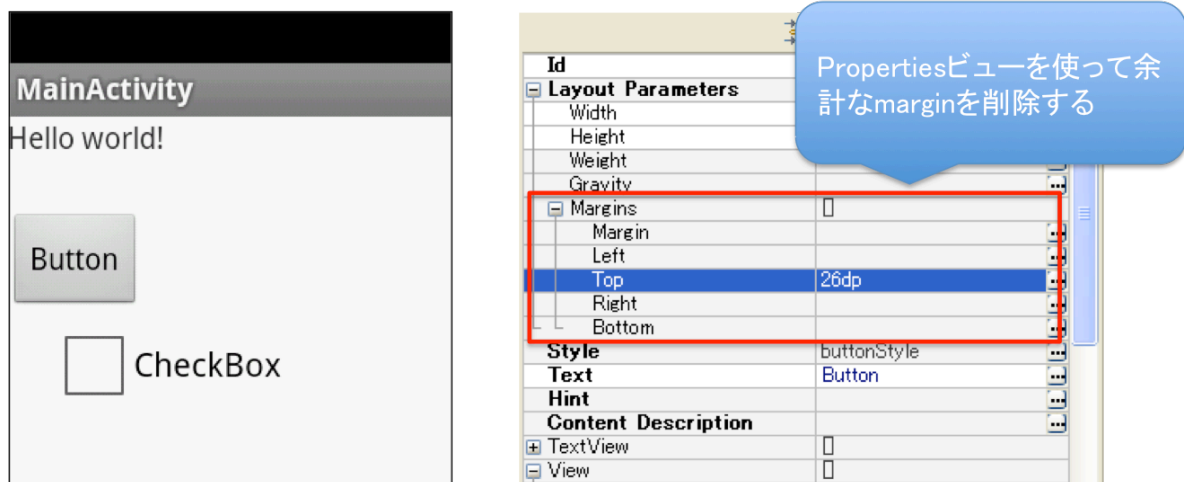
変更を保存し、アプリケーションを実行します。実行画面が次のようになっていることを確認します



余計な空白があって綺麗に整列されない場合

LinearLayout への変更時、配置情報によっては margin で補正されていることがあります。margin の値を消すことで対応出来ます。

[Properties View] で margin の設定値を確認を確認します。



4.6 EditText の追加

ボタン、チェックボックスはレイアウトエディタの GUI 機能を利用して追加しましたが、エディットテキストは、画面デザインリソースファイル（XML ファイル）を直接編集して追加します。

xml エディタを起動する

レイアウトエディタの activity_main.xml タブをクリックし、レイアウトエディタに XML ソースが表示されることを確認します。



EditText を追加する

EditText タグを activity_main.xml に追加します。

リスト 4.3: EditText を追加する

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ... />
3:
4:     <TextView
5:         ... />
6:
7:     <Button
8:         ... />
9:
10:    <CheckBox
11:        ... />
12:
13:    <EditText
14:        android:id="@+id/editText1"
15:        android:layout_width="wrap_content"
16:        android:layout_height="wrap_content"
17:        android:text="EditText" />
18:
19: </LinearLayout>
```

アプリケーションの実行

変更を保存し、アプリケーションを実行します。実行画面が図 4. のようになっていることを確認します



4.7 文字列リソース

アプリケーションで使う文字列をリソースファイルとして定義します。画面レイアウトを定義するファイルです。プロジェクトを新規作成すると、res/values フォルダに strings.xml という文字列リソースファイルが生成されます。

リスト 4.4: strings.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <resources>
3:
4:     <string name="app_name">HelloWorld</string>
5:     <string name="action_settings">Settings</string>
6:     <string name="hello_world">Hello world!</string>
7:
8: </resources>
```

文字列リソースは、`<string>` タグで定義します。name 属性には文字列リソースの名前、`<string>` タグの値に文字列を指定します。

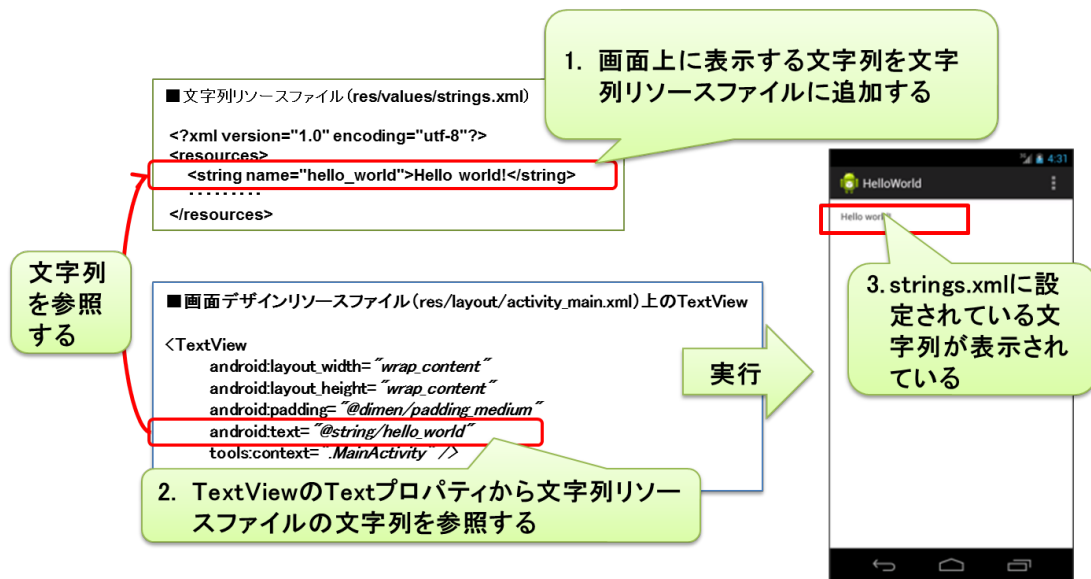
```
<string name="hello_world">Hello world!</string>
```

リソースファイルなどの XML ファイルから文字列リソースを参照する場合は、次のように記述します。

```
@<リソースの種類> / <リソース名>
```

初期状態の activity_main.xml の TextView の Text プロパティは次のように記述されています。

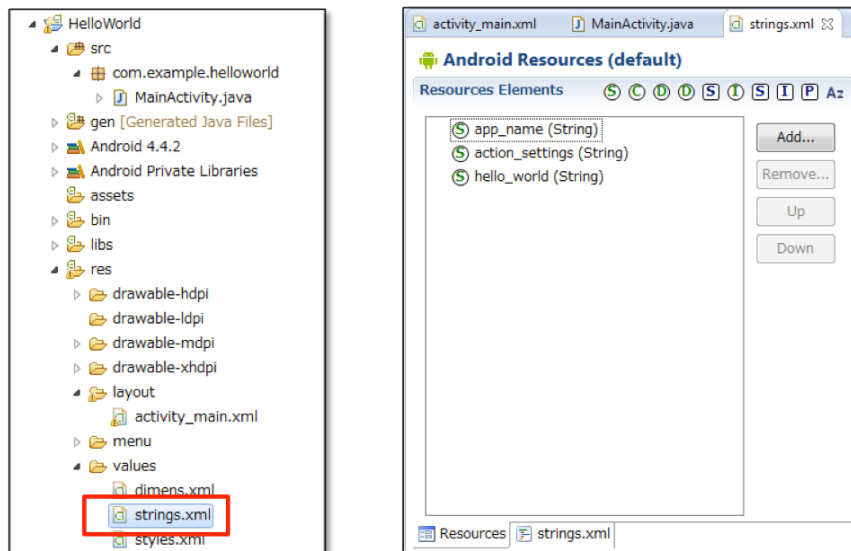
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

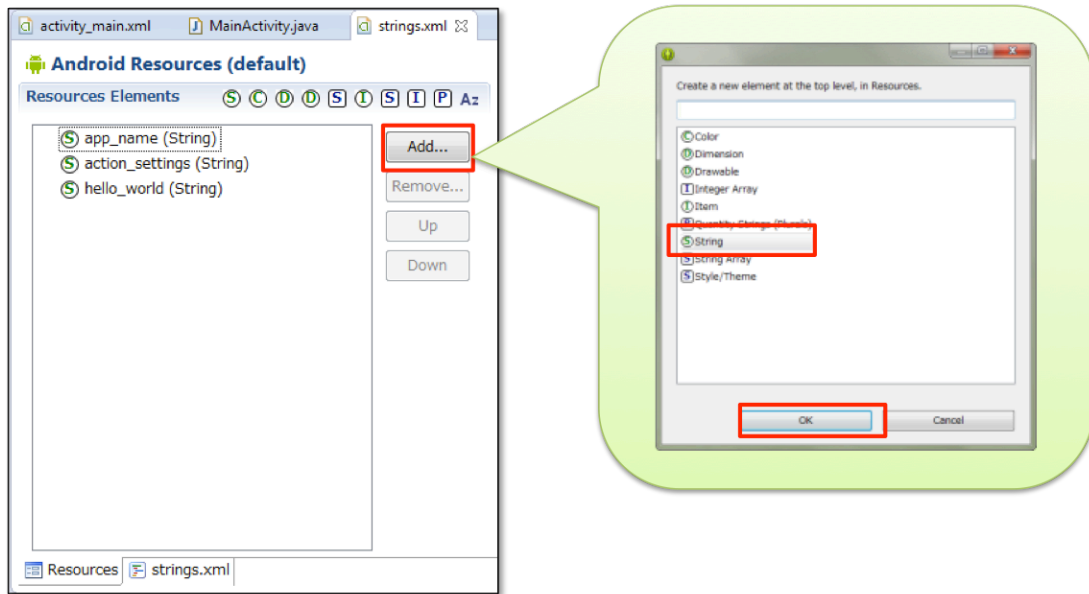
文字列リソースを追加する

文字列リソースファイルに文字列設定を追加し、Button の Text を Hello Button に変更します。

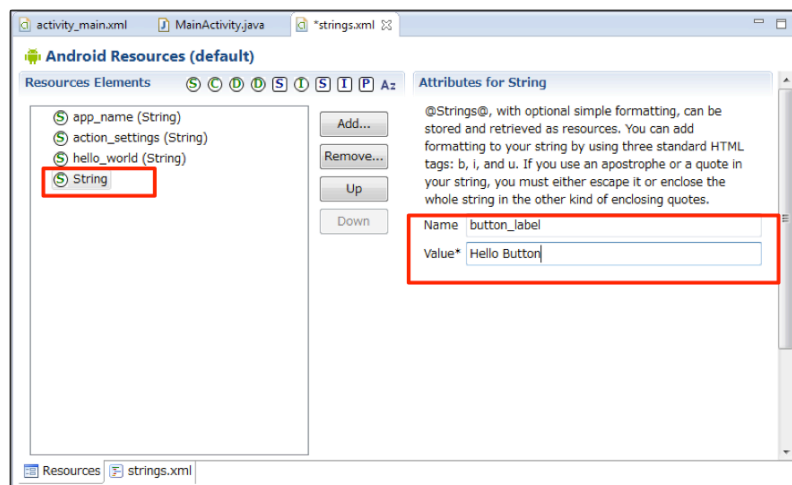
- Package Explorer から res/values/strings.xml をダブルクリックする
- res/values/strings.xml のファイルがリソースエディタで開かれ、右図の画面が表示されることを確認する



- Resources タブの [Add...] ボタンをクリックし、追加要素の選択画面を表示する
- 選択画面から、String を選択し、[OK] ボタンをクリックする



- Resources タブの Resources Elements に String が追加されていることを確認する
- Resources タブの Attributes for String の Name、Value に次の値を入力する
 - Name : button_label
 - Value : Hello Button

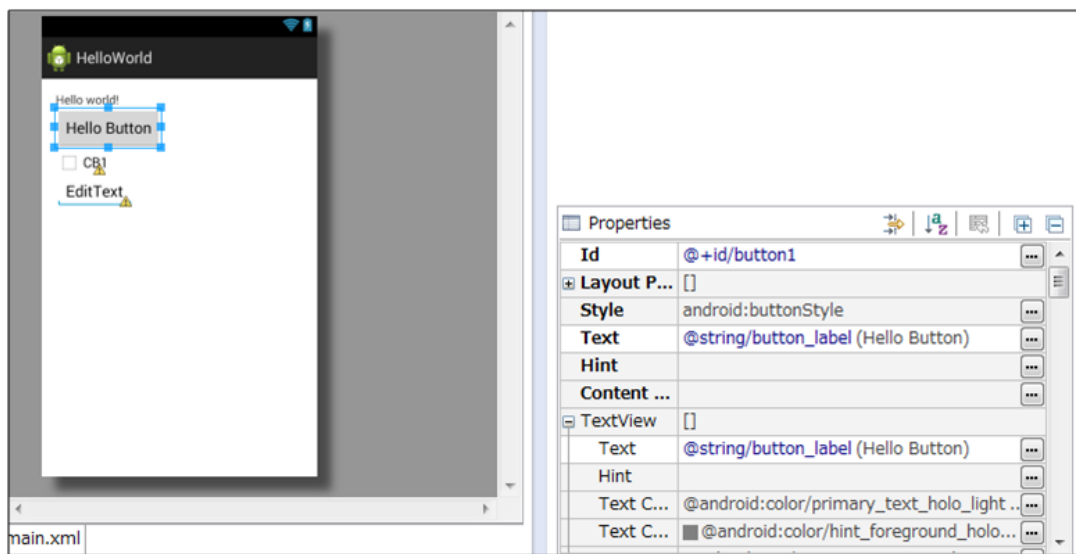


Button の Text を変更する

作成した文字列リソースを Button の Text プロパティに設定します。

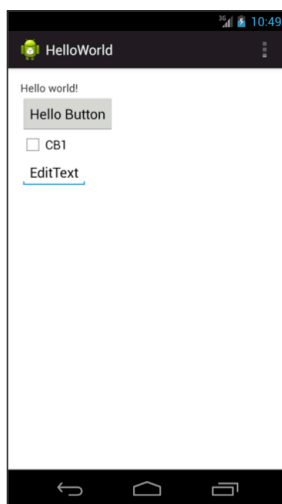
- Package Explorer から res/layout/activity_main.xml をダブルクリックする
- レイアウトエディタが起動したら、Layout タブを開いて Button01 を選択する

- Properties ビューから Text プロパティを @string/button_label に変更する



アプリケーションの実行

変更を保存し、アプリケーションを実行します。実行画面が次のようになっていることを確認します



4.8 AndroidManifest.xml

AndroidManifest.xml は、Android アプリケーションを設定するためのファイルで、XML で記述します。Android アプリケーションは必ず AndroidManifest.xml を持ちます。HelloAndroid

プロジェクトでは、次のような AndroidManifest.xml が生成されます。

リスト 4.5: AndroidManifest.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3:     package="com.example.helloworld"
4:     android:versionCode="1"
5:     android:versionName="1.0" >
6:
7:     <uses-sdk
8:         android:minSdkVersion="19"
9:         android:targetSdkVersion="19" />
10:
11:     <application
12:         android:allowBackup="true"
13:         android:icon="@drawable/ic_launcher"
14:         android:label="@string/app_name"
15:         android:theme="@style/AppTheme" >
16:         <activity
17:             android:name="com.example.helloworld.MainActivity"
18:             android:label="@string/app_name" >
19:             <intent-filter>
20:                 <action android:name="android.intent.action.MAIN" />
21:
22:                 <category android:name="android.intent.category.LAUNCHER" />
23:             </intent-filter>
24:         </activity>
25:     </application>
26:
27: </manifest>
```

<manifest>タグ

AndroidManifest.xml のルートタグで、必ず 1 つ記述します。複数記述することはできません。
<manifest>タグの子として<application>タグや<uses-sdk>タグなどを記述します。
<manifest>タグには次のようなプロパティを指定します。

- package : アプリケーションのパッケージ名
- android:versionCode : アプリケーションのバージョンを示す整数
- android:versionName : アプリケーションのバージョンを示す文字列 (ユーザ表示用)

<application>タグ

Android アプリケーションに関する情報を記述します。
<application>タグの子として<activity>タグ、<service>タグ、<provider>タグ、<receiver>タグなどを記述します。
<application>タグでは次のようなプロパティを指定します。

- android:description : アプリケーションの説明
- android:icon : アプリケーションのアイコン
- android:label : アプリケーションのラベル
- android:debuggable : ユーザモードでのデバッグの可否
- android:permission : アプリケーションの実行に必要なパーミッション
- android:theme : アプリケーションに適用するテーマ

<activity>タグ

Activity に関する情報を記述します。アプリケーションに複数の Activity が含まれている場合、Activity ごとに<activity>タグを記述します。

<activity>タグの子として<intent-filter>タグなどを記述します。

<activity>タグでは次のようなプロパティを指定します。

- android:name : Activity のクラス名
- android:icon : Activity のアイコン
- android:label : Activity のラベル

<intent-filter>タグ

Activity などのコンポーネントは Intent という非同期メッセージをやりとりします。<intent-filter>タグは、Activity がどの種類の Intent を受け取るかを指定します。子として<action>タグ、<category>タグ、<data>タグを記述します。

<uses-sdk>タグ

<manifest>タグの子として記述します。android:minSdkVersion 属性には、アプリケーションが必要とする Android のバージョンを API Level で指定します。

4.9 アプリケーションログの参照

android.util.Log クラスのメソッドを使って、Eclipse の [LogCat] ビューにメッセージを出力します。Log クラスでは次のメソッドを提供しています。

static int e(String tag, String msg):

LogCat] ビューに致命的なエラーの情報を出力する際に使います。

static int w(String tag, String msg):

LogCat] ビューに警告レベルのエラーの情報を出力する際に使います。

static int i(String tag, String msg):

LogCat] ビューにアプリケーション関連の情報を出力する際に使います。

static int d(String tag, String msg):

[LogCat] ビューにデバッグ情報を出力する際に使います。

static int v(String tag, String msg):

[LogCat] ビューに詳細な情報を出力する際に使います。

リスト 4.6: サンプルコード

```

1: Log.e("HelloWorld", "Error Message");           // エラーを出力するログ
2: Log.w("HelloWorld", "Warning Message");         // ワーニングを出力するログ
3: Log.i("HelloWorld", "Infomation Message");       // アプリケーション動作の情報を出力するログ
4: Log.d("HelloWorld", "Debug Message");           // デバッグ情報を出力するログ
5: Log.v("HelloWorld", "Verbose Message");         // 詳細情報を出力するログ

```

LogCat ビュー

Eclipse の [Window] メニューの [Show View] - [Other] を選択し、[LogCat] を選択すると、[LogCat] ビューが表示されます。

L...	Time	PID	TID	Application	Tag	Text
E	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Error Message
W	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Warning Message
I	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Infomation Message
D	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Debug Message
V	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Verbose Message

Hello World からログを出力する

HelloWorld の起動時にアプリケーションログを出力させましょう。

MainActivity.java を修正する

Package Explorer から MainActivity.java をダブルクリックし、エディタを起動させ次のように修正します。

リスト 4.7: MainActivity.java

```

1: @Override
2: protected void onCreate(Bundle savedInstanceState) {
3:     super.onCreate(savedInstanceState);
4:     setContentView(R.layout.activity_main);
5:
6:     Log.e("HelloWorld", "Error Message");
7:     Log.w("HelloWorld", "Warning Message");
8:     Log.i("HelloWorld", "Infomation Message");
9:     Log.d("HelloWorld", "Debug Message");
10:    Log.v("HelloWorld", "Verbose Message");
11:

```

```
12: }
```

アプリケーションの実行

変更を保存し、アプリケーションを実行します。LogCat ビューに5つのログが出力されていることを確認します。

L...	Time	PID	TID	Application	Tag	Text
E	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Error Message
W	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Warning Message
I	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Information Message
D	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Debug Message
V	07-19 03:01:45...	614	614	jp.oesf.tutorial	HelloWorld	Verbose Message

第5章

ユーザインタフェース 1

この章の目的

- View と ViewGroup の概要を理解する
- Toast を理解する
- OptionMenu を理解する
- AlertDialog を理解する

5.1 View

View とは

View とは、ボタンやチェックボックスなどの画面を構成するユーザインタフェースのことです。android.view.View クラスは Button クラスや TextView クラスなどの基底クラスです。Button や TextView などのクラスは android.widget パッケージに配置されているため、具体的なユーザインタフェースはウィジェットとも呼ばれます。また、複数の View をまとめる機能をもった View を ViewGroup と呼びます。

View の例

View には次のようなものがあります。(詳細については、後述します。)

TextView

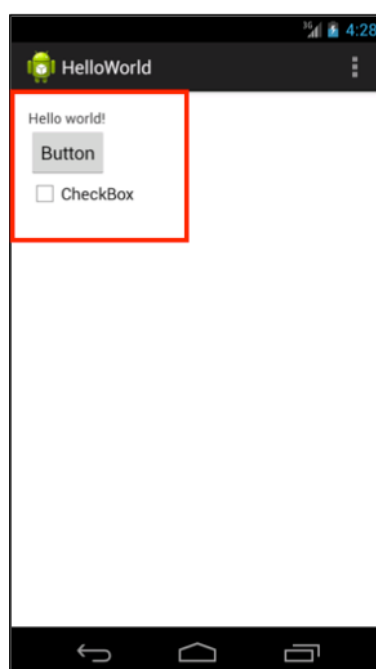
テキストを表示します。

Button

基本的な押しボタンです。ユーザのクリック操作に対応しています。

CheckBox

ON/OFF という状態をもっています。クリックすることで状態を変更することができます。



View のプロパティ

View をどのように表示するかを決定するのが、View のプロパティです。View のサイズや、表示する文字や画像や色など、余白をどのくらい使うかなどの情報をプロパティとして設定します。どんなプロパティが使えるかは View によって様々ですが、共通するプロパティも存在します。プロパティはレイアウトリソースの xml の属性で指定します。また、Eclipse にはプロパティを設定するためのプロパティビューが用意されています。

id

ビューの ID

layout_width

ビューの幅

layout_height

ビューの高さ

layout_margin

ビューの外側の余白

padding

ビューの内側の余白

リスト 5.1: プロパティの例

```
1: <Button
2:     android:id="@+id/button1"
```

```
3:         android:layout_width="wrap_content"
4:         android:layout_height="wrap_content"
5:         android:layout_alignLeft="@+id/textView1"
6:         android:layout_below="@+id/textView1"
7:         android:text="Button" />
```

5.2 View の作成

Eclipse のレイアウトエディタを使って、いろいろな View を作成してみましょう。

- TextView
- EditText
- Button
- CheckBox
- ImageView
- ProgressBar

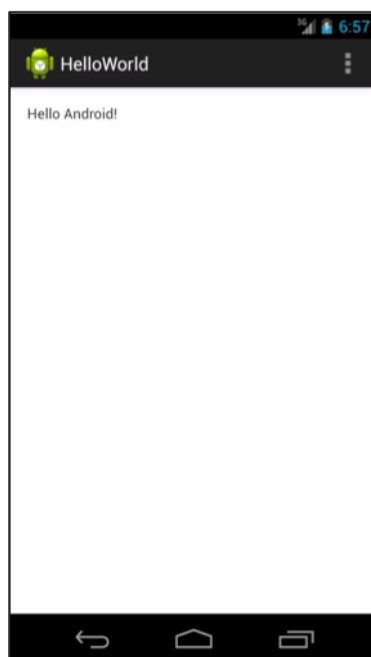
TextView

TextView とは

TextView は、テキスト（文字列）を表示するための View です。文字列のサイズや文字列のカラーなど変更する事が可能です。

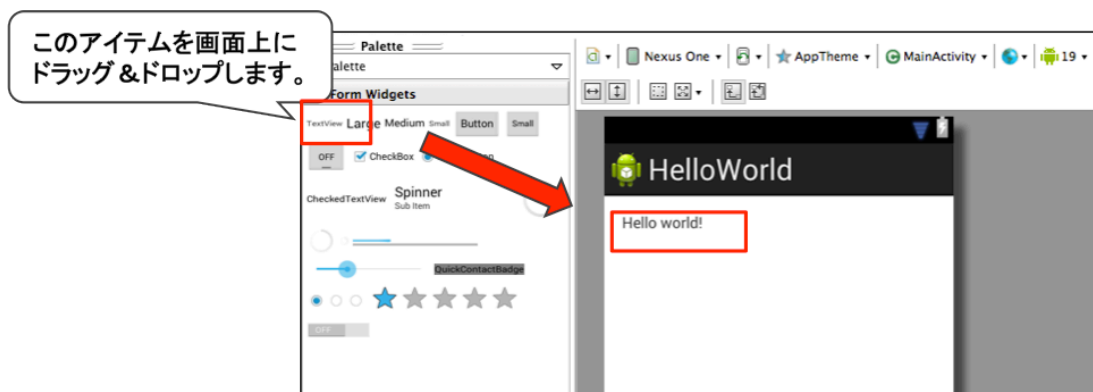
TextView の使い方

TextView を使って、「Hello Android!」と表示させてみましょう。



TextView の追加

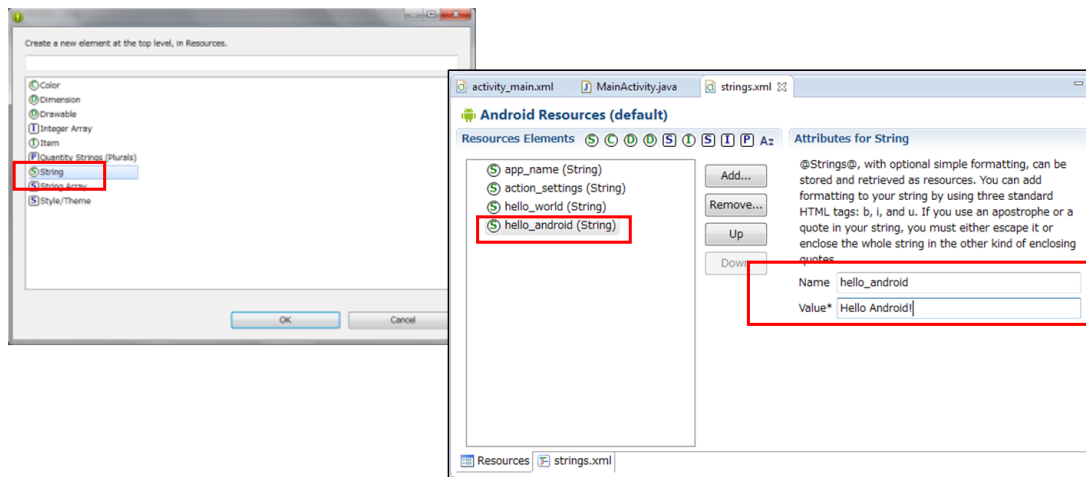
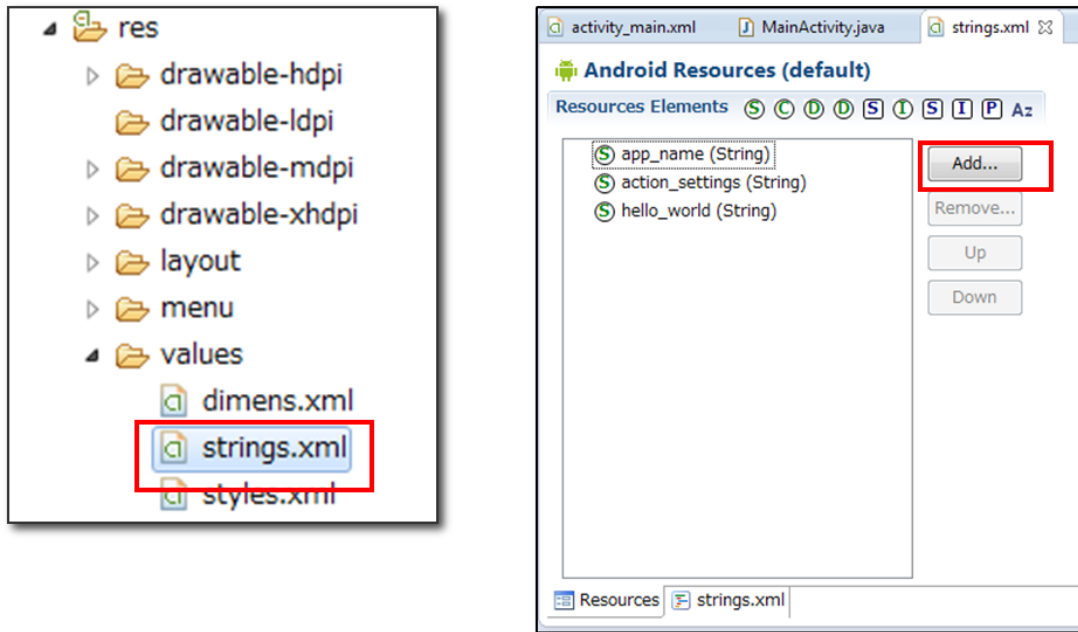
レイアウトファイルをエディタで開き、TextView を選択します。TextView を画面上にドラッグ & ドロップします。



String リソースの追加

res/values/strings.xml をエディタで開きリソースツール上から表示するテキストを追加します。

4章 デザイン以外のリソースを使用する参照



プロパティの変更

TextView に表示する文字列を変更します。プロパティの設定値を、次のように変更します。

text:

@string/hello_android

リスト 5.2: TextView の例

```
1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
```

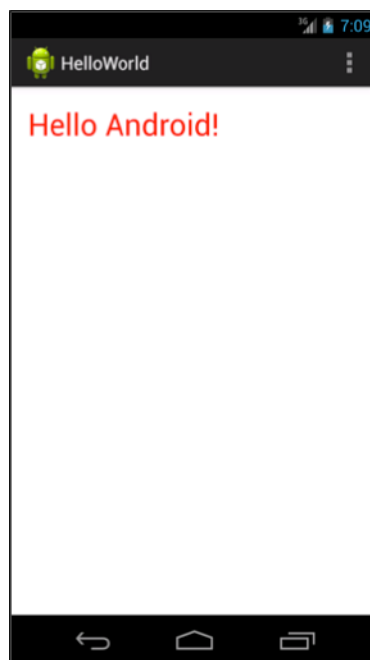
```
3:     >
4:
5:     <TextView
6:         android:id="@+id/textView1"
7:         android:layout_width="wrap_content"
8:         android:layout_height="wrap_content"
9:         android:text="@string/hello_android" />
10:
11: </RelativeLayout>
```

アプリケーションの実行

アプリケーションを実行し、TextView の値が「Hello Android!」になっていることを確認します。

TextView のフォントを変更する

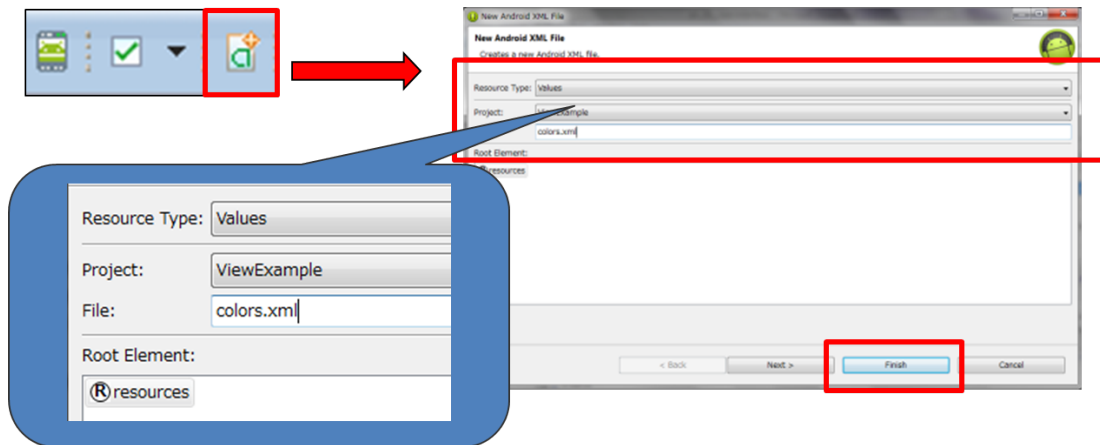
テキストサイズとテキストカラーを変更しましょう。



Color リソースファイルの作成

色を定義した colors.xml を次の手順で作成します。

1. /res/values を選択した状態で AndroidXML ファイル作成ボタンをクリックする。
2. ファイル名に「colors.xml」と指定し、Finish ボタンをクリックする
3. /res/values 以下に colors.xml が作成されていることを確認する



Color リソースの追加

次の手順で、カラーリソースを追加します。

1. res/values/colors.xml をダブルクリックしリソースエディタを起動する
2. 「add」ボタンをクリックし、一覧ウインドウから Color リソースを選択する
3. name の値を「hello_android_color」、Value の値を「#FF0000」に設定する

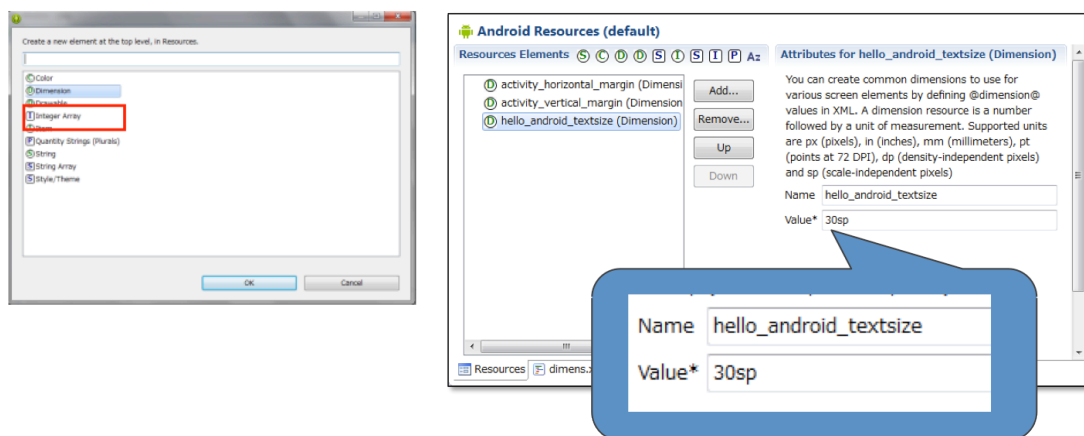


Dimension リソースの追加

次の手順で、Dimension リソースを追加します。

1. res/values/dimens.xml をダブルクリックしリソースエディタを起動する
2. 「add」ボタンをクリックし、一覧ウインドウから Dimension リソースを選択する
3. name の値を「hello_android_text_size」、Value の値を「30sp」*1に設定する

*1 Android ではテキストサイズの単位は sp で指定します



プロパティの変更

TextView に、テキストサイズとテキストカラーを設定します。プロパティの設定値を、次のように変更します。

textColor:

@color/hello_android_color

textSize:

@dimen/hello_android_text_size

リスト 5.3: TextView の例 2

```

1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView
6:         android:id="@+id/textView1"
7:         android:layout_width="wrap_content"
8:         android:layout_height="wrap_content"
9:         android:text="@string/hello_android"
10:        android:textColor="@color/hello_android_color"
11:        android:textSize="@dimen/hello_android_text_size" />
12:
13: </RelativeLayout>

```

アプリケーションの実行

アプリケーションを実行し、テキストカラーとテキストサイズが変更されていることを確認します。

EditText

EditText とは

EditText は、テキスト入力することができる View です。

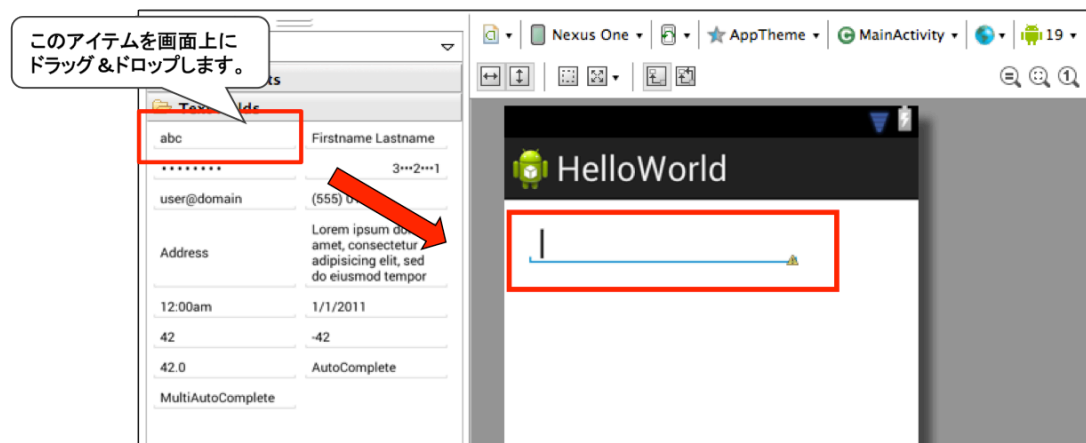
EditText の使い方

EditText を使って、「Hello Android!」と表示させてみましょう。



EditText の追加

レイアウトファイルをエディタで開き、EditText を選択します。EditText を画面上にドラッグ & ドロップします



プロパティの変更

EditView のプロパティの設定値を、次のように変更します。

text:

@string/hello_android

layout_width:

wrap_content

layout_height:

wrap_content

wrap_content と match_parent layout_width、layout_height は View の縦、横の長さを指定します。次のような値を指定できます。

wrap_content:

内容に応じて、適したサイズに設定される。

match_parent:

所属している ViewGroup のサイズの最大値が設定される。

数値 + 単位:

数値 + 単位でサイズを指定する。単位には dp を使うのが一般的。

リスト 5.4: EditText の例

```

1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <EditText
6:         android:id="@+id/editText1"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content"

```

```
9:         android:layout_alignParentLeft="true"
10:         android:layout_alignParentTop="true"
11:         android:text="@string/hello_android"
12:         android:ems="10" >
13:
14:         <requestFocus />
15:     </EditText>
16:
17: </RelativeLayout>
```

アプリケーションの実行

アプリケーションを実行し EditText が表示されていることを確認します。

Button

Button とは

基本的な押しボタンです。クリックイベントによって押したとき処理を実装することができます。基本的な押ボタン以外にも、ToggleButton や CheckBox など押した状態を管理するボタンも存在します。

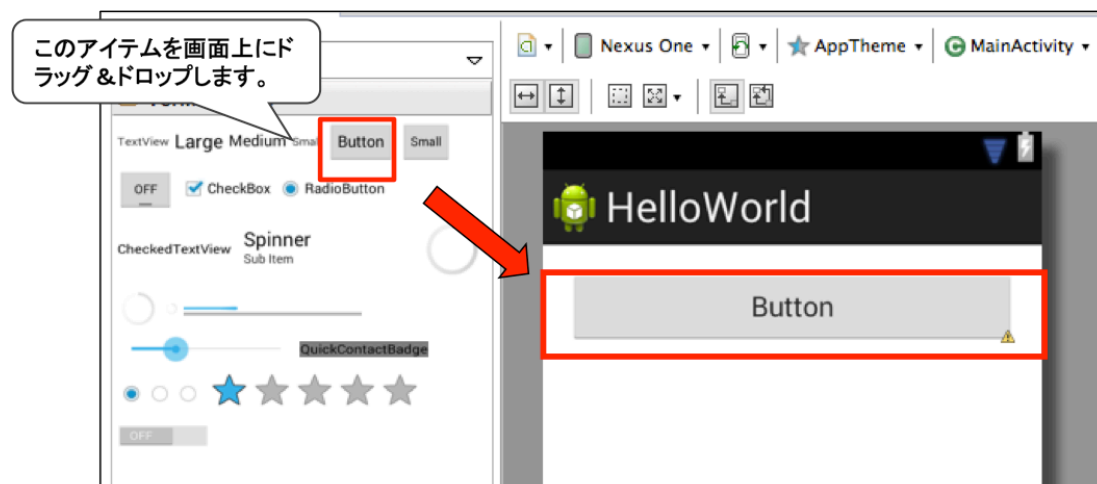
Button の使い方

Button を表示させてみましょう。



Button の追加

レイアウトファイルをエディタで開き、Button を選択します。Button を画面上にドラッグ&ドロップします



プロパティの変更

Button のプロパティの設定値を、次のように変更します。

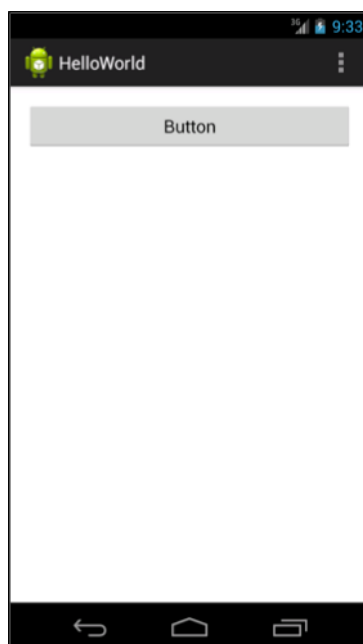
layout_width:

match_parent

layout_height:

wrap_content

アプリケーションの実行



リスト 5.5: Button の例

```
1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <Button
6:         android:id="@+id/button1"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content"
9:         android:layout_alignParentLeft="true"
10:        android:layout_alignParentTop="true"
11:        android:text="Button" />
12:
13: </RelativeLayout>
```

5.3 クリックイベント

作成したプロジェクトを実行し、ボタンをクリックしても何も起こりません。これはイベント処理を行っていないからです。Android でも、Java の GUI プログラミングと同じようにイベント処理を行う必要があります。

イベント処理の手順

Android でのイベント処理の手順は、Java の場合とほぼ同じです。

1. イベントリスナーインタフェースを実装するクラスを作成し、イベントハンドラメソッドにイベントに対応する処理を記述する。
2. イベントリスナーの実装クラスのオブジェクトを生成し、ボタンなどの View のオブジェクトに登録する。
3. ボタンのクリックなどのイベントが発生すると、登録したイベントリスナーのイベントハンドラメソッドがされる。

View.OnClickListener インタフェース

View へのクリックが発生すると、View.OnClickListener インタフェースの onClick メソッドが呼び出されます。onClick メソッドには引数としてイベントの発生元の View が渡されます。

クリックイベント対応

ボタンのクリックのイベント処理を追加してみましょう。ボタンがクリックされたときに、ボタンにクリックされた回数が表示されるようにします。具体的な手順は次のとおりです。

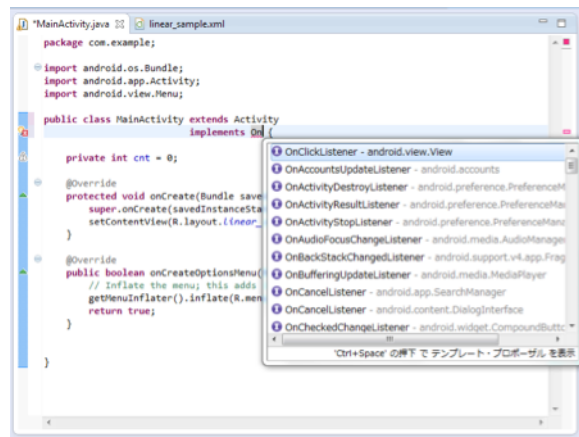
1. MainActivity クラスに View.OnClickListener インタフェースを実装する。
2. MainActivity クラスに onClick メソッドを追加し、ボタンクリック時の処理を記述する。
3. MainActivity クラスの onCreate メソッドに、setOnClickListener メソッドでイベントリスナーに登録する処理を記述する。

1. MainActivity クラスに View.OnClickListener インタフェースを実装する。

MainActivity クラスは次のように定義されています。

```
public class MainActivity extends Activity {
```

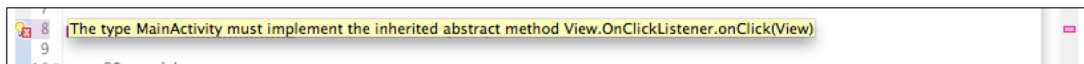
MainActivity.java を開き、extends Activity のうしろに「implements On」と入力して、Ctrl キー + スペースキーを押します。実装するインタフェースの候補が表示されるので、「OnClickListener - android.view.View」を選択します。



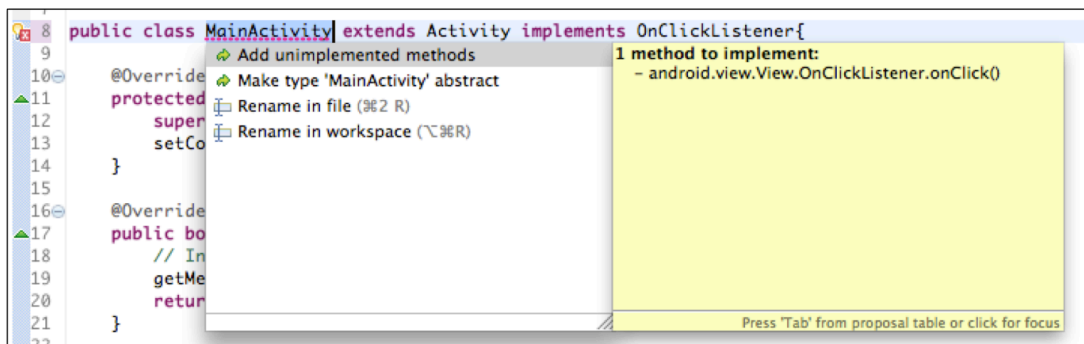
MainActivity クラスの定義の横には赤い×マークが付いています。これはソースコード内にエラーがあるという意味です。



この×マークにカーソルを合わせると、エラーの内容が表示されます。



さらに×マークをクリックすると、その対処方法が表示されます。



「Add unimplemented methods」を選択すると、MainActivity クラスに次のように onClick メソッドが追加されます。

```
1: @Override
2: public void onClick(View v) {
3:     // TODO Auto-generated method stub
```

```
4:  
5: }
```

2. MainActivity クラスに onClick メソッドを追加し、ボタンクリック時の処理を記述する

onClick メソッドの引数はイベントが発生した View です。この場合は、ボタンになります。ボタンは、レイアウトファイルで次のように定義されています。

```
<Button  
    android:id="@+id/button2"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentTop="true"  
    android:text="Button" />
```

ボタンは `android.widget.Button` クラスで表されます。MainActivity クラスに Button を保持する private フィールドを宣言します。"Button"と入力したところで `Ctrl + スペース` を押すと `"import android.widget"` が追加されます。

```
import android.widget.Button;  
  
public class MainActivity extends Activity implements OnClickListener{  
  
    private Button button;
```

Button クラスの `setText` メソッドを使えばボタンの文字列を設定できます。onClick メソッドに次のように記述して、Button の表示文字列を `"clicked!"` に設定します。

```
@Override  
public void onClick(View v) {  
    button.setText("clicked!");  
}
```

3. MainActivity クラスの onCreate メソッドに、setOnClickListener メソッドでイベントリスナを登録する処理を記述する。

MainActivity クラスの `onCreate` メソッドに、イベントリスナを登録するために、`setOnClickListener` メソッドの呼び出しを追加します。

```
public void setOnClickListener(OnClickListener l)
```

setOnClickListener メソッドの引数には View.OnClickListener インタフェースを実装したクラスのオブジェクトを渡します。今回の場合は、MainActivity クラス自体を表す this です。

イベントリスナを登録するのはボタンですが、ボタンはレイアウトファイルで定義されています。そのため、このレイアウトファイルから次のメソッドでボタンのオブジェクトを取得します。

```
public final View findViewById(int id)
```

findViewById メソッドには、レイアウトファイルで定義されているボタンのリソース ID を指定します。また、戻り値として View クラス型を返すため、Button 型にキャストします。

```
button = (Button)findViewById(R.id.button1);
```

取得した Button にイベントリスナを登録します。

```
button.setOnClickListener(this);
```

MainActivity クラスは次のようになります。

リスト 5.6: MainActivity.java

```
1: package com.example.chap05_view_examples;
2:
3: import android.os.Bundle;
4: import android.app.Activity;
5: import android.view.Menu;
6: import android.view.View;
7: import android.view.View.OnClickListener;
8: import android.widget.Button;
9:
10: public class MainActivity extends Activity implements OnClickListener{
11:
12:     private Button button;
13:     @Override
14:     protected void onCreate(Bundle savedInstanceState) {
15:         super.onCreate(savedInstanceState);
16:         setContentView(R.layout.activity_main);
17:
18:         button = (Button)findViewById(R.id.button1);
```



```
19:         button.setOnClickListener(this);
20:     }
21:
22:     @Override
23:     public boolean onCreateOptionsMenu(Menu menu) {
24:         // Inflate the menu; this adds items to the action bar if it is present.
25:         getMenuInflater().inflate(R.menu.main, menu);
26:         return true;
27:     }
28:
29:     @Override
30:     public void onClick(View v) {
31:         button.setText("clicked!");
32:     }
33:
34: }
```

onClick メソッドでクリックしたビューを判断する方法

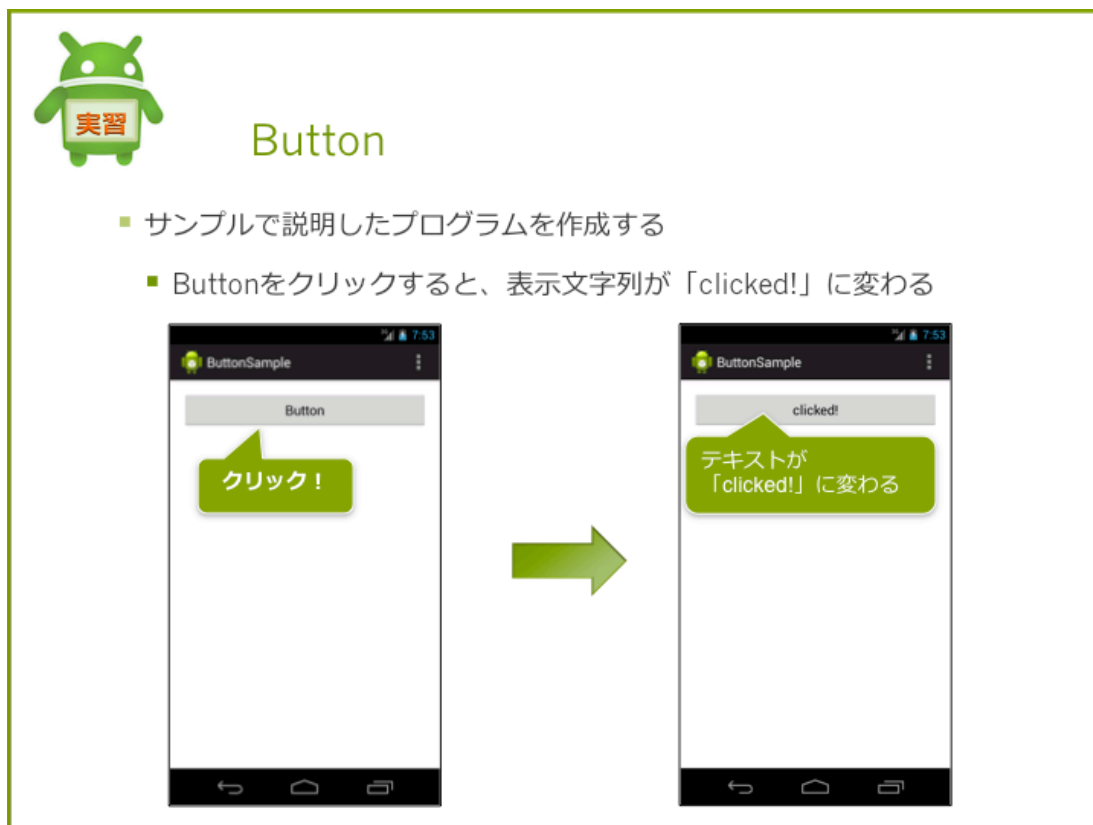
複数の Button に対してイベントリスナの登録を行うと、全ての Button のクリックイベントが、onClick メソッド集中します。

```
button1 = (Button)findViewById(R.id.button1);
button1.setOnClickListener(this);
button2 = (Button)findViewById(R.id.button2);
button2.setOnClickListener(this);
```

そのため、onClick メソッドで処理の振り分けが必要になります。処理を振り分けるには、引数「View」から、ビューの「id」を取得して判断します。次のコードは「android:id="@+id/button1"」がクリックされたことを判断する例です。

```
public void onClick(View v) {
    if(v.getId() == R.id.button1){
        // button1 がクリックされたときの処理
    }else if(v.getId() == R.id.button2){
        // button2 がクリックされたときの処理
    }
}
```

5.4 【実習】Button



実習

説明を参考にして、同じプログラムを作成してみましょう。

プロジェクト概要

表 5.1 プロジェクト概要

項目	設定値
Project Name	ButtonSample
Build Target	4.4
Aplication name	ButtonSample
Package	com.example.buttonsample
Create Activity	MainActivity

確認

アプリケーションを実行し、Button の表示文字列が「clicked!」になっていることを確認します。

解答

別ドキュメントを参照して下さい。

Button に OnClickListener を実装する方法

ここでは MainActivity クラスに View.OnClickListener インタフェースを実装しましたが、その他の方法を紹介します。

匿名クラスを使って OnClickListener を実装する

匿名の内部クラスを使って実装することができます。

1. Button#setOnClickListener で引き数内で OnClickListener を生成
2. onClick メソッドをオーバーライドする

```
Button bt = (Button)findViewById(R.id.button1);
    bt.setOnClickListener(new OnClickListener() {    . . . 1

    @Override
    public void onClick(View v) {
        // クリックされたときの処理    . . . 2
    }
});
```

レイアウトファイルに onClick を設定する

レイアウトファイルに OnClickListener を設定することができます。

1. View の onClick プロパティに、クリック時に呼び出したいメソッド名を設定します。ここでは onClickButton を指定します。

```
<Button
    android:id="@+id/button1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:onClick="onClickButton"
    android:text="Button" />
```

1. Activity に onClickButton メソッドを追加します。引数には View を指定します。

```
public void onClickButton(View v) {
    button.setText("clicked!");
}
```

```
}
```

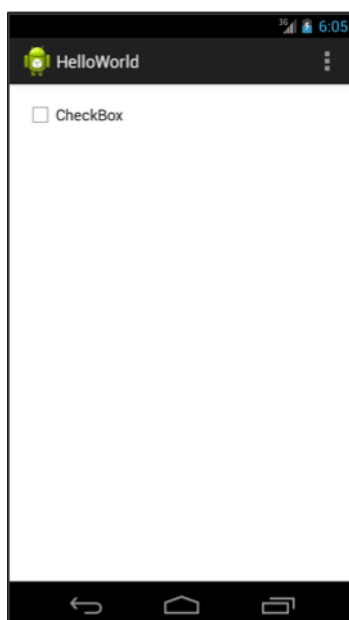
CheckBox

CheckBox とは

CheckBox は選択状態を持つことができる View です。CheckBox をタップすると選択状態を変更することができます。

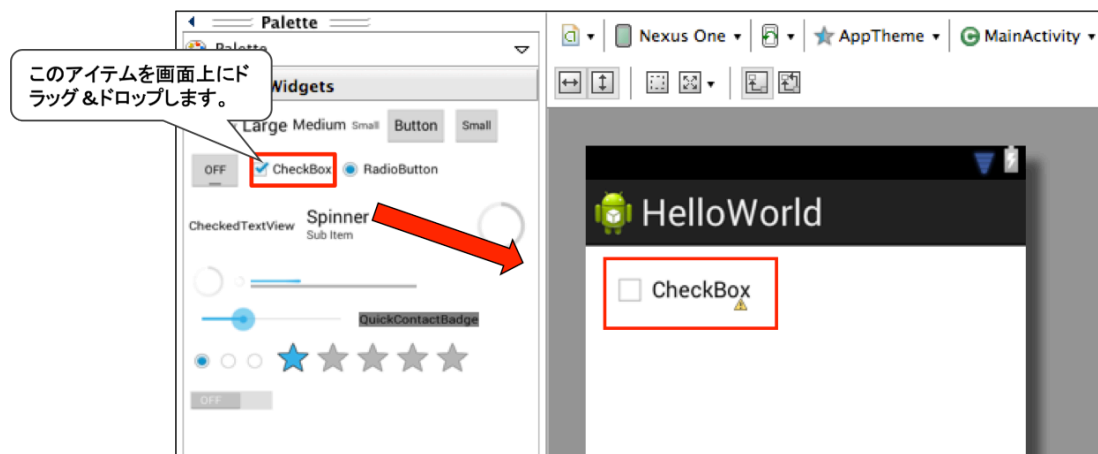
CheckBox の使い方

CheckBox を使って、「Hello Android!」と表示させてみましょう。



CheckBox の追加

レイアウトエディタを使って CheckBox を配置します。



プロパティの変更

CheckBox のプロパティの設定値を、次のように変更します。

text:

@string/hello_android

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    >

    <CheckBox
        android:id="@+id/checkBox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="CheckBox" />

</RelativeLayout>
```

アプリケーションの実行

アプリケーションを実行し CheckBox が表示されていることを確認します。CheckBox をクリックするとチェック状態が変わることも確認します。

チェック状態の変更通知を受け取る方法

CheckBox のチェック状態の変更通知を受け取るには、OnCheckedChangeListener を使用します。

onCheckedChanged メソッドの引数 isChecked を使用してチェック状態を取得します。onClick-

Listener の場合と違って、ユーザがクリックしなくても、チェック状態が変更されると呼び出されます。

```
checkBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {  
  
    @Override  
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
        if (isChecked) {  
            // チェックされた状態の時の処理を記述  
        } else {  
            // チェックされていない状態の時の処理を記述  
        }  
    }  
});
```

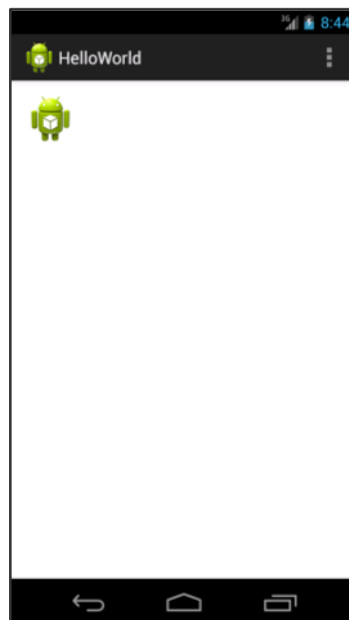
ImageView

ImageView とは

ImageView は画像を表示することができる View です。

ImageView の使い方

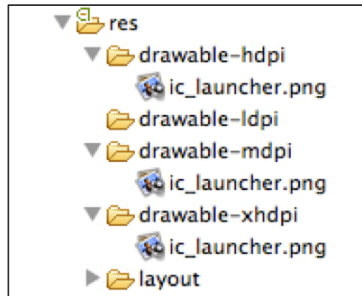
ImageView を使って、ic_launcher.png を表示させてみましょう。



画像リソースの準備

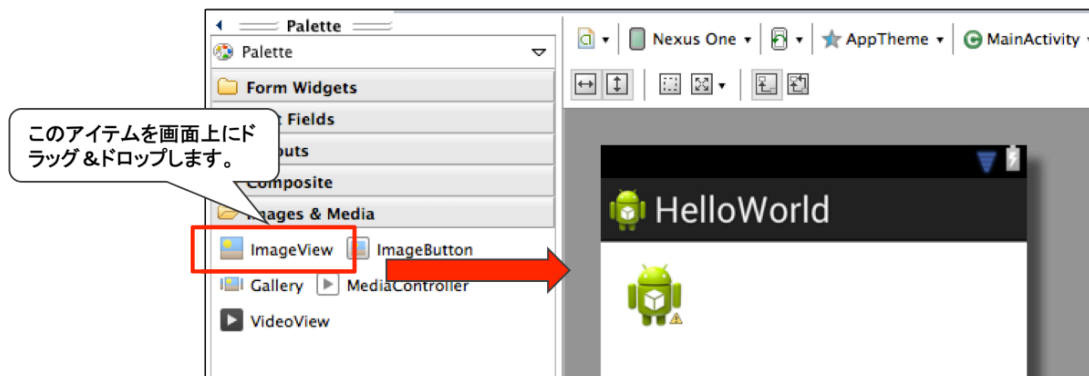
表示させたい画像を `res/drawable(-hdpi, -xhdpi など)` フォルダ内に用意します (エクスプローラからドラッグアンドドロップで配置できます)。

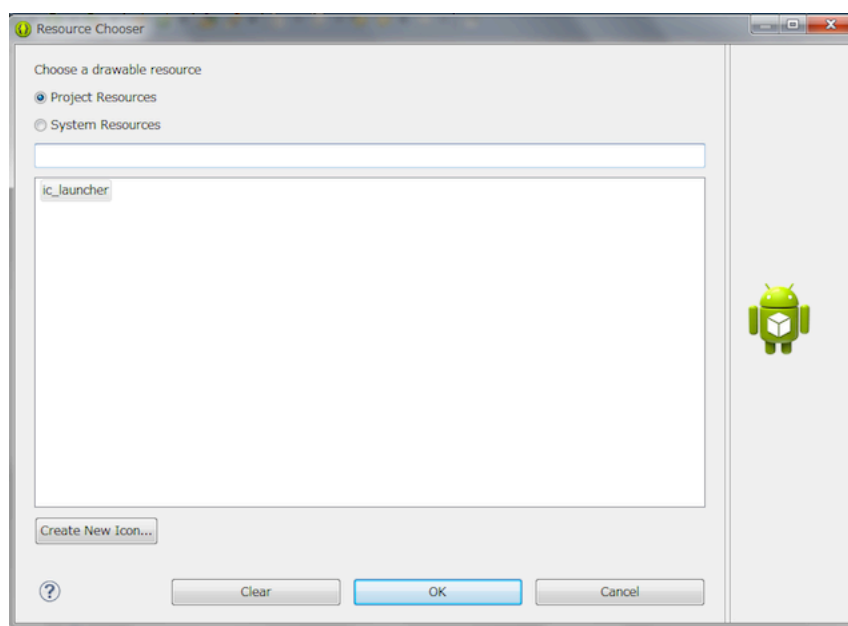
デフォルトで、アプリケーションのランチャーアイコンが保存されています。



ImageView の追加

レイアウトエディタを使って `ImageView` を配置します。配置すると、自動的に Resource Chooser ウィンドウが表示されます。一覧から `ic_launcher` を選択します。





プロパティの変更

ImageView の src プロパティが選択した画像リソースに設定されていることを確認します。

src:

@drawable/ic_launcher

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:src="@drawable/ic_launcher" />

</RelativeLayout>
```

アプリケーションの実行

アプリケーションを実行し ImageView が表示されていることを確認します。

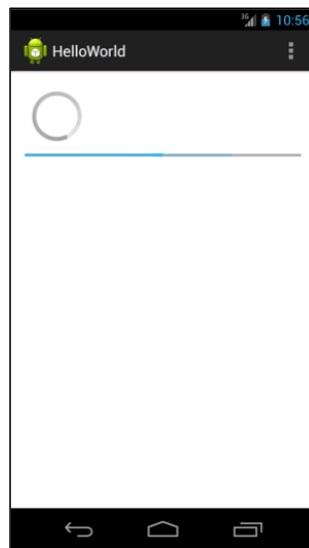
ProgressBar

ProgressBar とは

ProgressBar は、処理の進捗状況を表示するための View です。回転タイプのものと、進捗バータイプのものなど、見た目を変更することができます。

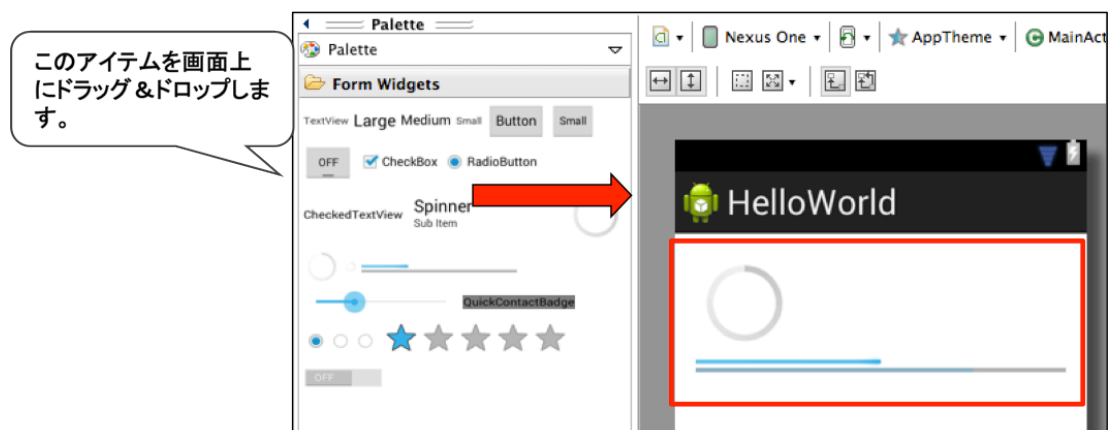
ProgressBar の使い

次のような2種類の ProgressBar を表示させてみましょう。



ProgressBar の追加

レイアウトエディタを使って ProgressBar を配置します。2種類の ProgressBar を表示させてみましょう。



プロパティの変更

ProgressBar のプロパティの設定値を、次のように変更します。

progressBar1 変更なし

progressBar2

style:

?android:attr/progressBarStyleHorizontal

progress:

50

max:

100

secondaryProgress:

75

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    >

    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

    <ProgressBar
        android:id="@+id/progressBar2"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:progress="50"
        android:max="100"
        android:secondaryProgress="75"
        android:layout_alignLeft="@+id/progressBar1"
        android:layout_below="@+id/progressBar1" />

</RelativeLayout>
```

アプリケーションの実行

アプリケーションを実行し ProgressBar が表示されていることを確認します。

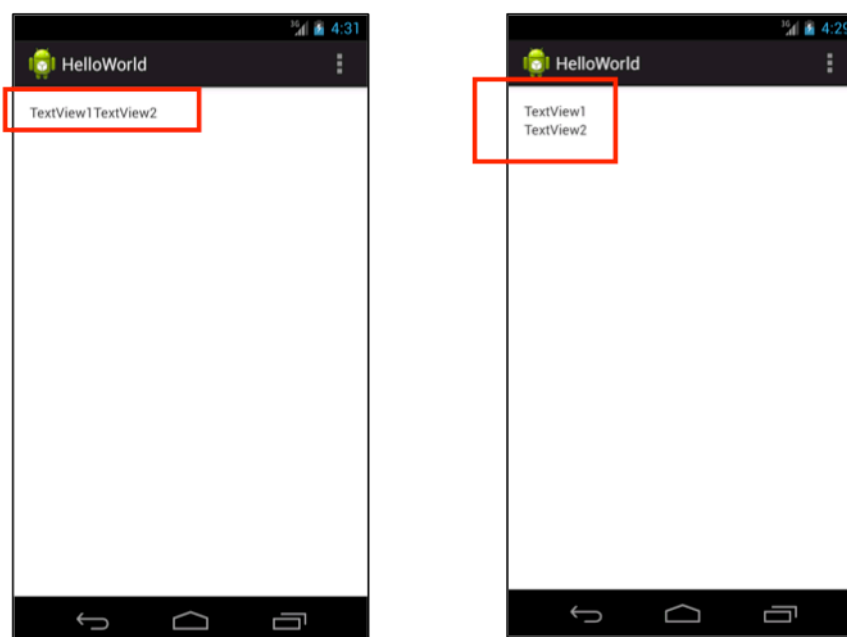
5.5 ViewGroup

ViewGroup とは

ViewGroup は内部に TextView などの他の View を含むことができる View です。ViewGroup の内部に他の ViewGroup を含むこともできます。

ViewGroup の例

図 5. の例では、LinearLayout を使って、内部に TextView を配置しています。



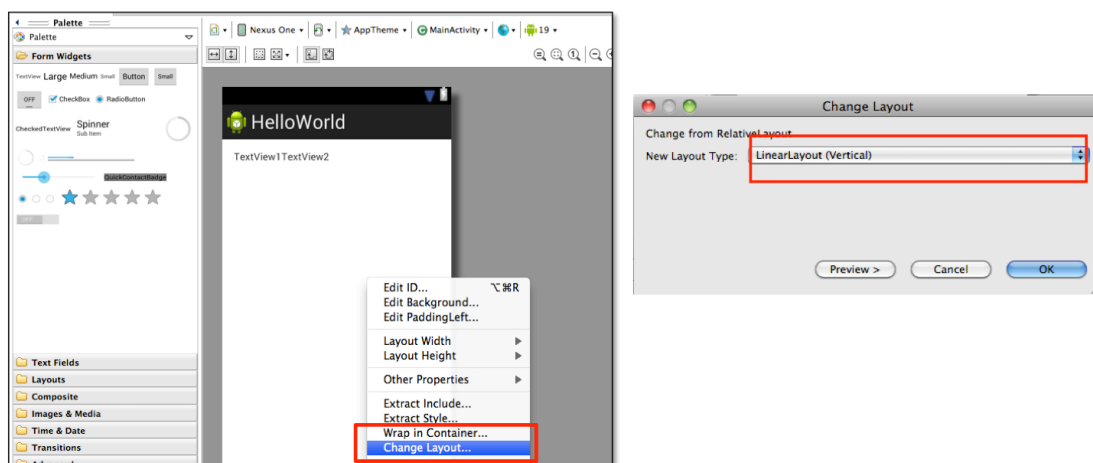
LinearLayout

LinearLayout とは

LinearLayout は View を垂直方向または水平方向に配置することができる ViewGroup です。

LinearLayout の使い方

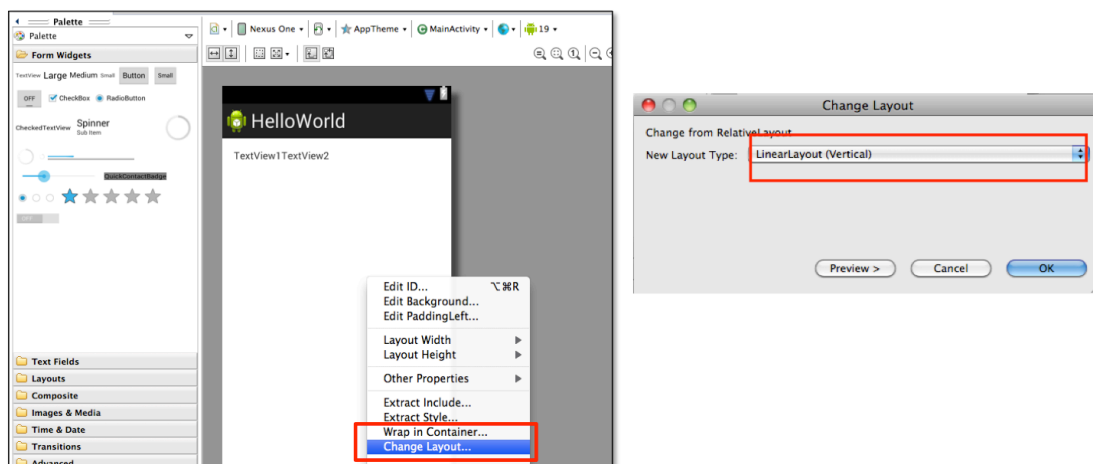
2 つの TextView を水平方向に配置してみましょう。それぞれの TextView には「TextView1」「TextView2」と表示させます。



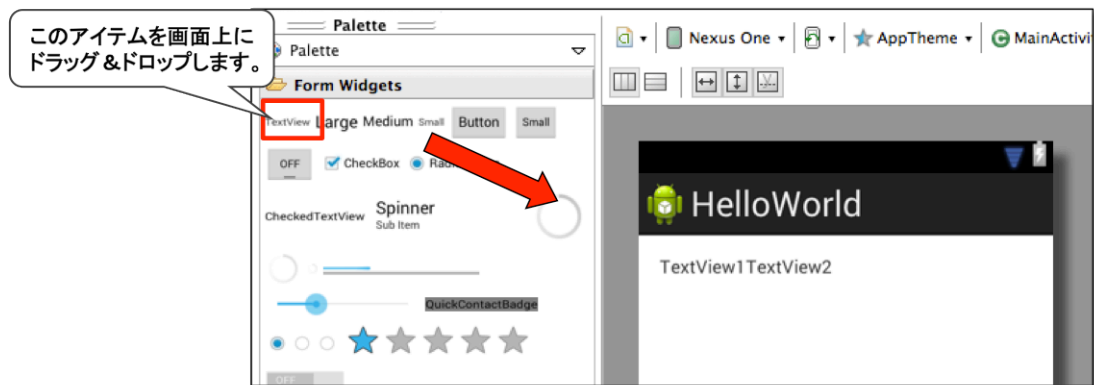
LinearLayout の追加

デフォルトは RelativeLayout になっている為、LinearLayout に変更します。

1. プレビュー画面上で右クリックし、[Change Layout] を選択
2. [New Layout Type] を [LinearLayout(Horizontal)] に変更



TextView を 2 つ追加します。



プロパティの変更

[New Layout Type] で [LinearLayout(Horizontal)] を選択すると、orientation プロパティの値が horizontal に設定されます。

orientation プロパティは、View の配置する方向を設定します。vertical を指定すると、垂直方向に配置され、horizontal を指定すると、水平方向に配置されます。

orientation:

horizontal

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView1" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView2" />

</LinearLayout>
```

アプリケーションの実行

アプリケーションを実行し TextView が横並びで表示されていることを確認します。

FrameLayout

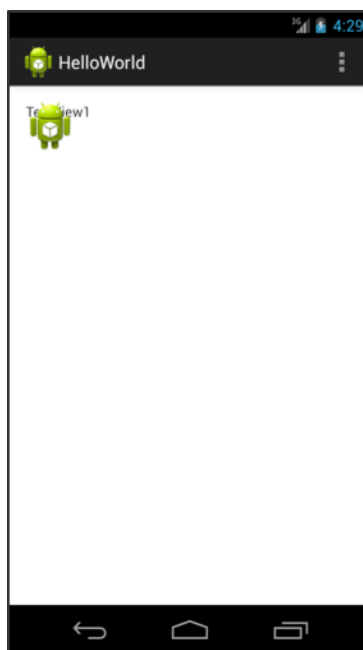
FrameLayout とは

FrameLayout は一つの View を配置することを目的としたレイアウトです。FrameLayout に追加した View はデフォルトでは、親となる FrameLayout の左隅（座標 (0.0) の位置に表示）に配置されます。複数の View を配置した場合も親となる FrameLayout の左隅の位置に表示されるので配置した View が重なって表示されます。

- FrameLayout のメリット
 - View 同士を重ねることができます
- FrameLayout のデメリット
 - View を整列させることが難しい

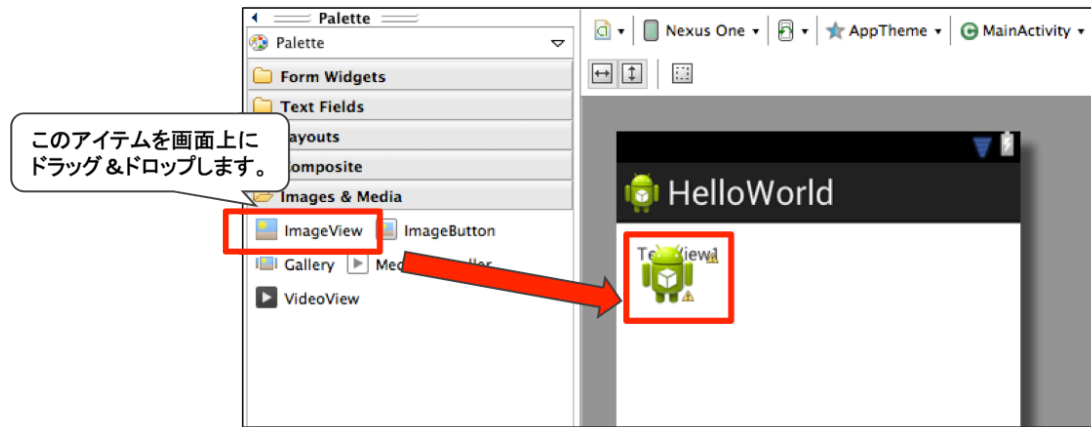
FrameLayout の使い方

FrameLayout に TextView と ImageView を配置してみましょう。



FrameLayout の追加

プレビュー画面上で右クリックし、[Change Layout] を選択し [FrameLayout] に変更し、TextView と ImageView を追加します。



プロパティの変更

変更なし

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/FrameLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView1" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />

</FrameLayout>
```

アプリケーションの実行

アプリケーションを実行し TextView と ImageView が重なって表示されていることを確認します。

RelativeLayout

RelativeLayout とは

RelativeLayout は View(親)の位置を決め、その位置を元に View(子)の位置を相対的(Relative)に指定することができる ViewGroup です。相対的に参照される View(親)は 参照する View(子)よりも先に定義する必要があります。

RelativeLayout のメリット 基準となる View(親)の位置を変更すると View(親)を基準として配置した View(子)の位置も自動的に変更されます。

RelativeLayout のデメリット 基準となっている View(親)を削除することができなくなります。

RelativeLayout のプロパティ

RelativeLayout は、次のようなプロパティを使って、View を相対的に配置します。

layout_centerInParent:

説明: 指定した View を画面の中心に親として配置する

layout_above:

@+id/center

layout_alignLeft:

基準となる View の上に表示する

layout_below:

基準となる View の下に表示する

layout_toLeftOf:

基準となる View の左に表示する

layout_toRightOf:

基準となる View の右に表示する

layout_alignLeft:

指定した View の左側の境界に合わせて整列する

layout_alignRight:

指定した View の右側の境界に合わせて整列する

layout_alignTop:

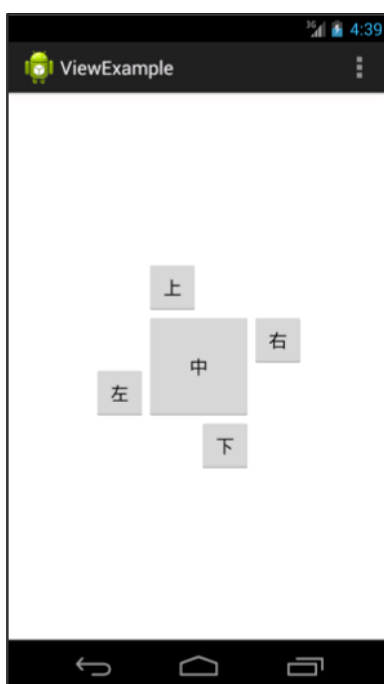
指定した View の上側の境界に合わせて整列する

layout_alignBottom:

指定した View の下側の境界に合わせて整列する

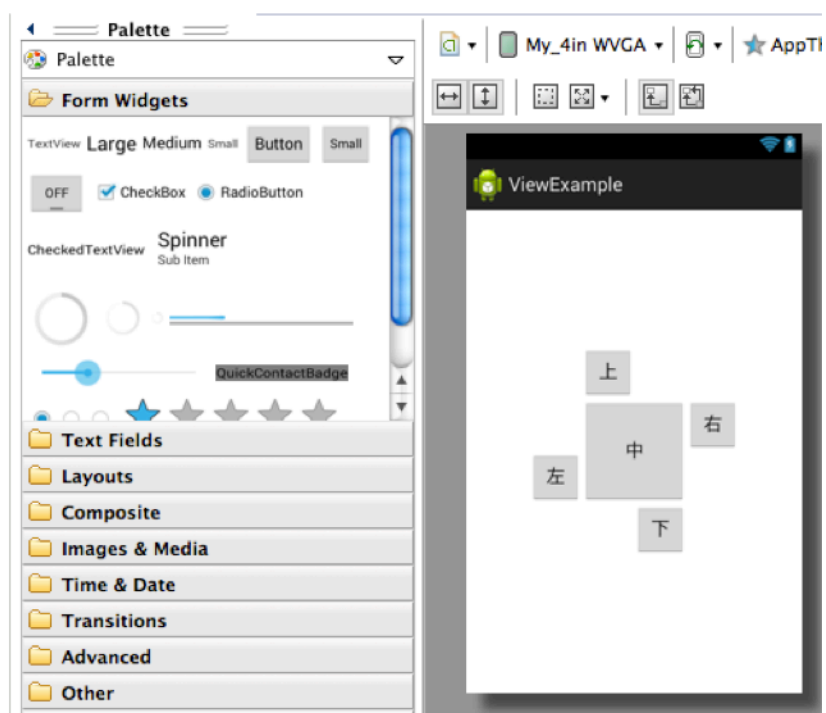
RelativeLayout の使い方

RelativeLayout を使って View を配置してみましょう。



RelativeLayout の追加

RelativeLayout プロジェクトを新規作成した直後の状態で、既に activity_main.xml に追加されています。



Button を配置する 次の手順で、Button を 5 つ配置しましょう。

1. 基準となる中ボタンを配置する
2. 中ボタンを基準に上ボタンと下ボタンを追加する
3. 中ボタンを基準に左ボタン、右ボタンをを追加する
4. 各 Button のテキスト文字列、上、中、下、左、右をリソースで定義する
5. 文字列リソースを、各 Button の Text プロパティに設定する

プロパティの変更

自動追加されるコードは親との位置関係と、自由な座標を補正するための margin (余白) が設定されています。親 View が正しいことを確認し、margin が設定されていないことを確認します。

中ボタン

id:

@+id/center

layout_width:

100dp

layout_height:

100dp

layout_centerInParent:

true 説明：指定した View を画面の中心に親として配置する

text:

@string/center

上ボタン

id:

@+id/avobe

layout_width:

50dp

layout_height:

50dp

layout_above:

@+id/center

layout_alignLeft:

@+id/center

text:

@string/avobe

下ボタン

```
id:
    @+id/below
layout_width:
    50dp
layout_height:
    50dp
layout_below:
    @+id/center
layout_alignRight:
    @+id/center
text:
    @string/below
```

左ボタン

```
id:
    @+id/left
layout_width:
    50dp
layout_height:
    50dp
layout_toLeftOf:
    @+id/center
layout_alignBottom:
    @+id/center
text:
    @string/left
```

右ボタン

```
id:
    @+id/right
layout_width:
    50dp
layout_height:
    50dp
layout_toRightOf:
    @+id/center
layout_alignTop:
    @+id/center
text:
```

@string/right

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    ...
>

<Button
    android:id="@+id/center"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_centerInParent="true"
    android:text="@string/center" />

<Button
    android:id="@+id/below"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_below="@+id/center"
    android:layout_alignRight="@+id/center"
    android:text="@string/below" />

<Button
    android:id="@+id/above"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_above="@+id/center"
    android:layout_alignLeft="@+id/center"
    android:text="@string/above" />

<Button
    android:id="@+id/left"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_toLeftOf="@+id/center"
    android:layout_alignBottom="@+id/center"
    android:text="@string/left" />

<Button
    android:id="@+id/right"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_toRightOf="@+id/center"
    android:layout_alignTop="@+id/center"
    android:text="@string/right" />

</RelativeLayout>
```

アプリケーションの実行

アプリケーションを実行し、次のことを確認しましょう。

- 画面中央に中ボタン

- 中ボタンの上に左揃えで上ボタン
- 中ボタンの下に右揃えで下ボタン
- 中ボタンの左に下揃えで左ボタン
- 中ボタンの右に上揃えで右ボタン

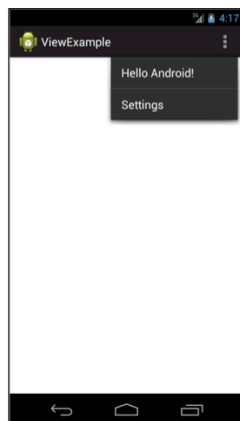
5.6 OptionMenu

OptionMenu とは

「MENU」ボタンを押すことで表示されるメニューです

OptionMenu の使い方

OptionMenu に「Hello Android!」と表示させてみましょう。さらに、Hello Android!が選択された時にログを出力させましょう。



Lev	Time	PID	TID	Application	Tag	Text
V	12-08 04:19:53.204	2907	2907	com.example.viewe...	MainActivity	Hello Androidが選択されました。

menu リソースの修正

OptionMenu 用のリソースファイルは、プロジェクト作成時に「res/menu/」フォルダ以下にファイル名「main.xml」で用意されています。main.xml ファイルに Hello Android! メニューを追加します。

1. main.xml をダブルクリックし、リソースエディタを起動する
2. Add ボタンをクリックし、一覧から Item を選択する
3. 新規メニューに次の値を設定する

id:

@+id/item1

Title:`@string/hello_android`

main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"/>
    <item
        android:id="@+id/item1"
        android:title="@string/hello_android">
    </item>

</menu>
```

MainActivity の修正

MainActivity にはメニューを表示する処理と、メニューが選択された時の処理を追加します

メニューを表示する処理を追加する

Menu ボタンを押した時に OptionMenu を表示する処理を追加します。Menu ボタンを押すと、Activity の onCreateOptionsMenu メソッドが呼び出されます。このメソッドをオーバーライドし、OptionMenu の表示処理を追加します。このメソッドは、プロジェクト作成時に既に追加されています。

```
public class MainActivity extends Activity {

    ...

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

メニューが選択された時の処理を追加する

表示されたメニューを選択すると、onOptionsItemSelected メソッドが呼び出されます。引数には選択された MenuItem が渡されます。onOptionsItemSelected メソッド内では、MenuItem の getItemId メソッドを使って、選択されたメニューに合わせて処理を振り分けることができます。

```
public class MainActivity extends Activity {  
  
    ...  
  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) {  
        if(item.getItemId() == R.id.item1){  
            Log.v("MainActivity", "Hello Android!が選択されました。");  
            return true;  
        }  
        return super.onOptionsItemSelected(item);  
    }  
}
```

アプリケーションの実行

アプリケーションを実行し、次のことを確認しましょう。

- メニューボタンをクリックした時に、Hello Android!というメニューが表示されている
- Hello Android!メニューを選択すると、「Hello Android!が選択されました」というログが出力される

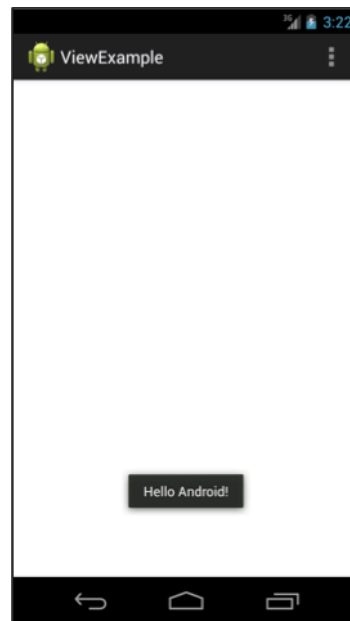
5.7 Toast

Toast とは

Toast は短いメッセージを数秒間表示します。メッセージは数秒後に自動的に消えます。Toast が表示されている間は画面を操作可能です。

Toast の使い方

Toast を使って「Hello Android!」と表示させてみましょう。



Toast オブジェクトを作成する Toast クラスに定義されている `makeText` メソッドを使って、Toast オブジェクトを作成します。

リスト 5.7: `makeToast` メソッドの定義

```
1: public static Toast makeToast (Context context, CharSequence text, int duration)
2: public static Toast makeToast (Context context, int resId, int duration)
```

引数の説明

context:

コンテキスト

text:

表示させるテキスト

resId:

表示させるテキストのリソース ID

duration:

表示時間。 `Toast.LENGTH_SHORT` (短時間) か `Toast.LENGTH_LONG` (長時間) を指定する。

Toast を表示する

`show` メソッドを実行すると Toast が表示されます。

```
public void show ()
```

MainActivity の修正

onCreate に Toast を表示する処理を追加します。

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Toast.makeText(this, R.string.hello_android, Toast.LENGTH_SHORT).show();  
    }  
}
```

アプリケーションの実行

アプリケーションを実行し、Toast が表示されていることを確認しましょう。

5.8 AlertDialog

AlertDialog とは

メッセージとボタンを表示することができるダイアログボックスです

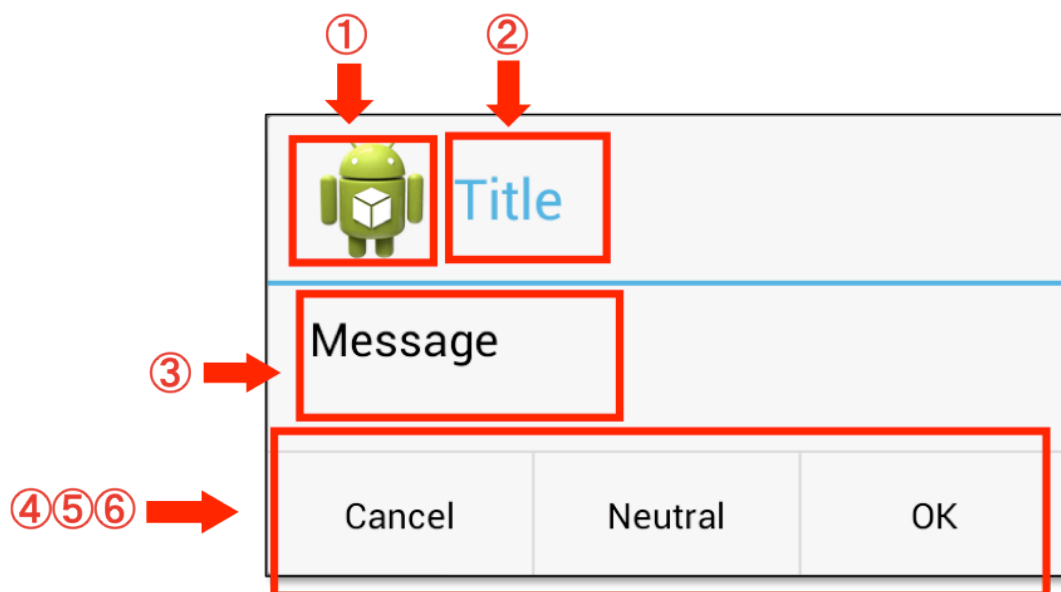
AlertDialog の使い方

AlertDialog を使ってみましょう。

AlertDialog の特徴

AlertDialog では、次の UI で構成されています。

1. アイコン
2. Title テキスト
3. MessageText
4. PositiveButton
5. NeutralButton
6. NegativeButton



AlertDialog を表示する

AlertDialog は AlertDialog.Builder を使って表示します。表示内容に対応するデータのセッターメソッドを使用して Dialog のどこに何を表示するかを設定します。各ボタンの設定は DialogInterface.OnClickListener を使ってボタンのクリックイベントを取得します。

表 5.2 プロジェクト概要

#	メソッド (引数省略)
1	setIcon
2	setTitle
3	Message
4	setPositiveButton
5	setNegativeButton
6	setNeutralButton

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        AlertDialog.Builder builder = new AlertDialog.Builder(this);

        // アイコン
        builder.setIcon(R.drawable.ic_launcher);
    }
}
```

```
// タイトル
builder.setTitle("Title");

// メッセージ
builder.setMessage("Message");

// クリックイベントの設定
builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(MainActivity.this, R.string.hello_android,
            Toast.LENGTH_LONG).show();
    }
});
builder.setNegativeButton("Cancel", null);
builder.setNeutralButton("Neutral", null);

// AlertDialog を表示します。
builder.show();
}
}
```

5.9 まとめ課題



アプリケーション概要

この章で学習した内容のまとめ実習です。次のようなアプリケーションを作成しましょう

動作手順

1. アプリケーションを起動する
2. メニュー「Alert」を表示する
3. Alert を選択すると、AlertDialog を表示する。
4. ボタンをクリックすると、Toast を表示し、AlertDialog が閉じる

表 5.3 Toast に表示するメッセージ

ボタン	メッセージ
OK	OK
Cancel	CANCEL

解答

解答ドキュメントを参考して下さい。

第6章

画面遷移

この章の目的

- 画面遷移の仕方を理解する
- 画面間での連携方法を理解する

6.1 シンプルな画面遷移

これまでは、1つの画面だけのアプリケーションを作成してきましたが、多くのアプリケーションは複数の画面で構成されています。そのためには Android で画面遷移するための仕組みを知っておく必要があります。

まずは、新しい画面を起動するだけの、シンプルな画面遷移アプリケーションを作成してみましょう。

画面遷移する方法

画面遷移とは Activity というコンポーネントを起動することです。コンポーネントは Intent を使って起動することができます。Activity から Activity を呼び出すには、呼び出し元の Activity で、Intent オブジェクトを作成し、起動する Activity の情報を指定します。作成した Intent オブジェクトを使用して Activity を呼び出すメソッドを実行すると、遷移先画面が表示されます。新たに作成した画面へ遷移する場合は、遷移先画面の Activity を AndroidManifest.xml に登録する必要があります。



図: 画面 A から画面 B へ画面遷移するイメージ

Intent を使用して、画面遷移を行う

プログラムでは次の手順で、画面遷移の処理を行います。

1. Intent オブジェクトを作成する
Intent の引数に、遷移元 Activity のオブジェクト、遷移先 Activity のクラス情報を与えて生成します。
2. startActivity メソッドを実行する
メソッドの引数には 1. で作成した Intent オブジェクトを指定します。

リスト 6.2: MainActivity から NextActivity に遷移する例

```
1: import android.app.Activity;
2: import android.view.View;
3: import android.view.View.OnClickListener;
4: import android.content.Intent;
5:
6: public class MainActivity extends Activity implements OnClickListener {
7:
8:     ... 略...
9:     public void onClick(View view) {
10:         Intent intent = new Intent(this,NextActivity.class);    ... 1
11:         startActivity(intent);    ... 2
12:     }
13: }
```


AndroidManifest.xml の設定に追加する

追加した Activity は AndroidManifest.xml の設定に追加する必要があります。AndroidManifest.xml の `<application>` タグの中に Activity 情報を追加します。

リスト 6.2: NextActivity を AndroidManifest.xml に追加する例

```
1: <application
2:     ... >
3:     ...
4:     <activity
5:         android:name="com.example.viewexample.NextActivity"
6:         android:label="@string/title_activity_next" >
7:     </activity>
8: </application>
```

6.2 【実習】画面遷移 1



実習 1

メイン画面の「next」ボタンを押した時、次の画面へ遷移するアプリケーションを作成しましょう。

プロジェクト概要

表 6.1 プロジェクト概要

項目	設定値
Project Name	ActivitySample
Build Target	4.4
Aplication name	ActivitySample
Package	com.example.activitysample
Create Activity	MainActivity

手順

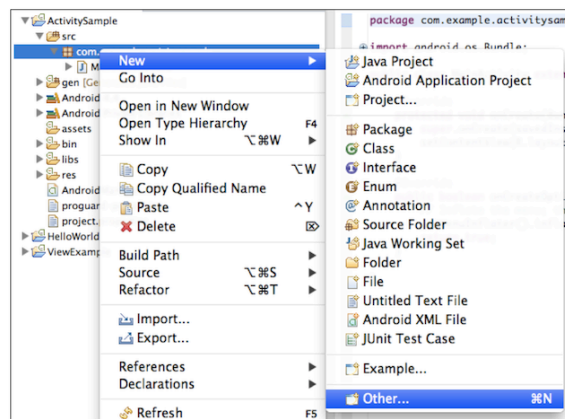
1. 遷移先画面の Activity を追加する

2. 遷移先画面のレイアウトを修正する
3. MainActivity のレイアウトを修正
4. MainActivity に画面遷移処理の追加する

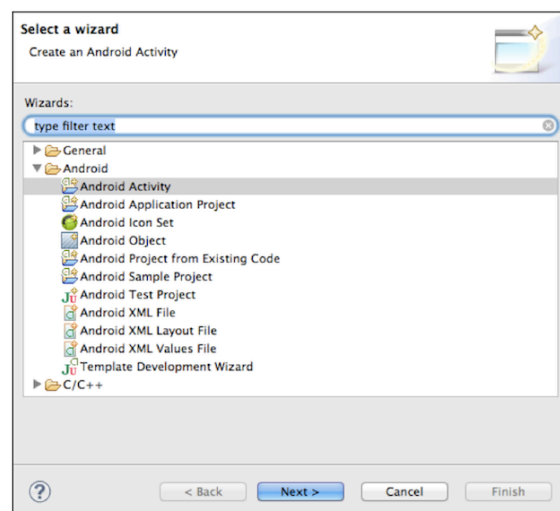
手順 1. 遷移先画面の Activity を追加する

ADT には、簡単に Activity の追加する機能が用意されています。この機能を使うと、Activity クラスの作成、リソースファイルの作成および AndroidManifest.xml の登録まで自動的に実装されます。

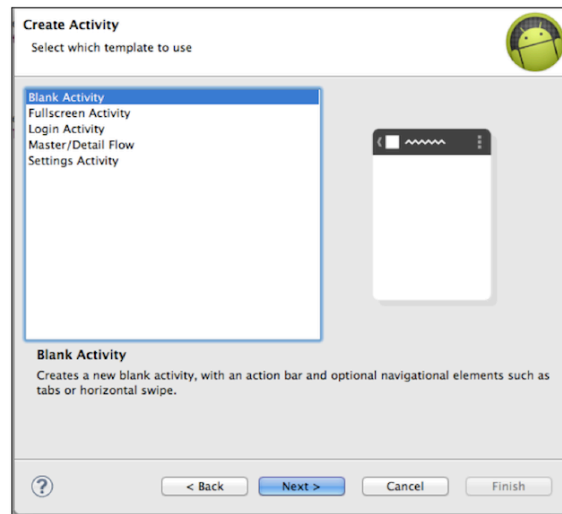
- パッケージツリーの `com.example.activitysample` を右クリックし、[New] > [Other] を選択します。



- Select a wizard 画面で [Android] > [Android Activity] を選択します。



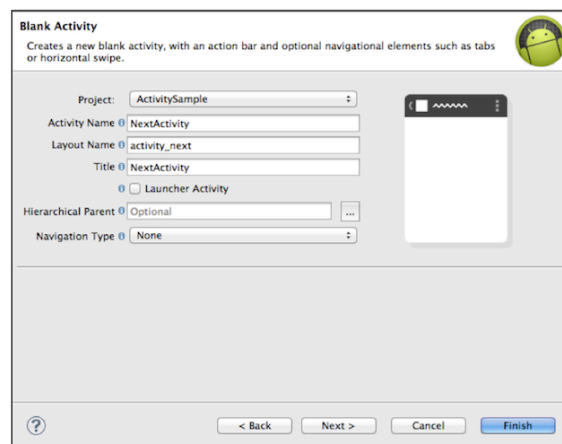
- Create Activity 画面で [Blank Activity] を選択します。



- Blank Activity 画面で 以下のように設定し、 [Finish] をクリックします。

表 6.2 Blank Activity の設定

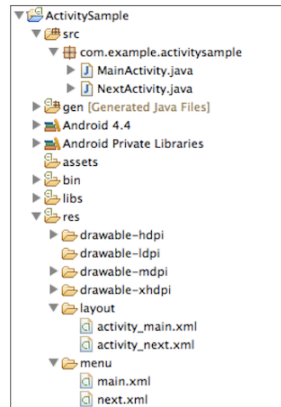
項目	設定値
Project	NextActivity
Layout Name	activity_next
Title	NextActivity
Luncher Activity	チェックしない
Hierarchical Parent	未入力
Navigation Type	None



自動作成されたファイルと内容の確認

ADT の機能を使って新規作成されたファイルと追加コードを確認します。

- プロジェクトに次のファイルが追加されていることを確認します。
 - NextActivity.java
 - activity_next.xml
 - next.xml



- Activity は画面表示のための基本的なコードが追加されています。

リスト 6.3: NextActivity.java の初期コード

```
1: package com.example.activitysample;
2:
3: import android.os.Bundle;
4: import android.app.Activity;
5: import android.view.Menu;
6:
7: public class NextActivity extends Activity {
8:
9:     @Override
10:    protected void onCreate(Bundle savedInstanceState) {
11:        super.onCreate(savedInstanceState);
12:        setContentView(R.layout.activity_next);
13:    }
14:
15:    @Override
16:    public boolean onCreateOptionsMenu(Menu menu) {
17:        // Inflate the menu; this adds items to the action bar if it is present.
18:        getMenuInflater().inflate(R.menu.next, menu);
19:        return true;
20:    }
21:
22: }
```

- strings.xml には NextActivity の文字が設定された、title_activity_next リソースが作成されています。

リスト 6.4: strings.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <resources>
3:
4:     <string name="app_name">ActivitySample</string>
5:     <string name="action_settings">Settings</string>
6:     <string name="hello_world">Hello world!</string>
7:     <string name="title_activity_next">NextActivity</string>     この文字列が追加され
                        ている
8:
9: </resources>
```

- AndroidManifest.xml の<application>タグに NextActivity が追加されていることを確認します。

リスト 6.5: AndroidManifest.xml の確認

```
1:     <activity
2:         android:name="com.example.activitysample.NextActivity"
3:         android:label="@string/title_activity_next" >
4:     </activity>
```

手順 2. 遷移先画面のレイアウトを修正する

NextActivity の画面デザインを次のように修正します。

リスト 6.6: activity_next.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:id="@+id/LinearLayout1"
4:     android:layout_width="match_parent"
5:     android:layout_height="match_parent"
6:     android:orientation="vertical"
7:     android:paddingBottom="@dimen/activity_vertical_margin"
8:     android:paddingLeft="@dimen/activity_horizontal_margin"
9:     android:paddingRight="@dimen/activity_horizontal_margin"
10:    android:paddingTop="@dimen/activity_vertical_margin"
11:    tools:context=".NextActivity" >
12:
13:    <TextView
14:        android:id="@+id/text_message"
15:        android:layout_width="wrap_content"
```

```
16:         android:layout_height="wrap_content"
17:         android:text="@string/title_activity_next" />
18:
19: </LinearLayout>
```

手順 3. MainActivity のレイアウトを修正

MainActivity の画面デザインを次のように修正します。

リスト 6.7: activity_main.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:id="@+id/LinearLayout1"
4:     android:layout_width="match_parent"
5:     android:layout_height="match_parent"
6:     android:orientation="vertical"
7:     android:paddingBottom="@dimen/activity_vertical_margin"
8:     android:paddingLeft="@dimen/activity_horizontal_margin"
9:     android:paddingRight="@dimen/activity_horizontal_margin"
10:    android:paddingTop="@dimen/activity_vertical_margin"
11:    tools:context=".MainActivity" >
12:
13:    <Button
14:        android:id="@+id/button1"
15:        android:layout_width="match_parent"
16:        android:layout_height="wrap_content"
17:        android:onClick="onClickNextButton"
18:        android:text="@string/next" />
19:
20: </LinearLayout>
```

strings.xml Button の表示テキストに使用するための、文字列リソースを追加します。

```
<string name="next">Next</string>
```

手順 4. MainActivity に画面遷移処理の追加する

ボタンをクリックすると onClickNextButton メソッドが呼び出されます。このメソッドに画面遷移の処理を追加します。Intent オブジェクトを作成し、第1引数に Activity のオブジェクト、第2引数に遷移先の class を指定し、startActivity メソッドを実行します。

リスト 6.8: onClickNextButton メソッド

```
1: public void onClickNextButton(View v){  
2:     Intent intent = new Intent(this, NextActivity.class);  
3:     startActivity(intent);  
4: }
```

確認

アプリケーションを実行し、MainActivity の Next ボタンをクリックすると画面遷移することを確認します。

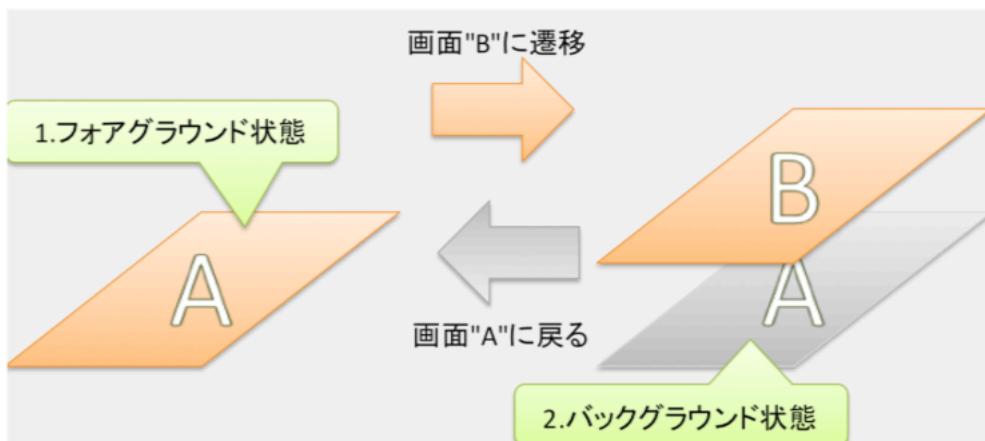


解答

解答ドキュメントを参照して下さい。

6.3 遷移元の画面に戻る

遷移した画面を終了すると、遷移前の画面が表示されます。これはフォアグラウンド状態の Activity が破棄されて、バックグラウンド状態だった Activity がフォアグラウンド状態になるためです。



Activity の終了方法

バックキーを押すと Activity は破棄され、直前の Activity に戻りますが、プログラムから Activity を終了することもできます。

Activity を終了するには、Activity の finish メソッドを呼び出します。finish メソッドを呼ぶと、フォアグラウンドの Activity は onDestroy メソッドが呼び出され破棄されます。フォアグラウンドの Activity が破棄されると、直前の Activity が再表示されます。

```
finish();
```

6.4 【実習】画面遷移 2



実習 2

前回の実習で作成したアプリケーションに、NextActivity の終了機能を追加しましょう。

手順

1. NextActivity のレイアウトを修正する
2. NextActivity に終了処理を追加する

手順 1. NextActivity のレイアウトを修正する

NextActivity の画面デザインを次のように修正します。

NextActivity のレイアウト

リスト 6.9: activity_next.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView
6:         ...
7:     />
8:     <Button
9:         android:id="@+id/button1"
10:        android:layout_width="match_parent"
11:        android:layout_height="wrap_content"
12:        android:onClick="onClickFinishButton"
13:        android:text="@string/finish" />
14:
15: </LinearLayout>
```

strings.xml

Button の表示テキストに使用するための、文字列リソースを追加します。

```
<string name="finish">Finish</string>
```

手順 2. NextActivity に終了処理を追加する

Finish ボタンがクリックされたときの処理を追加します。

リスト 6.10: onClickFinishButton メソッド

```
1: public void onClickFinishButton(View v){
2:     finish();
3: }
```

確認

アプリケーションを実行し、NextActivity で Finish ボタンをクリックすると画面が終了することを確認する。



解答

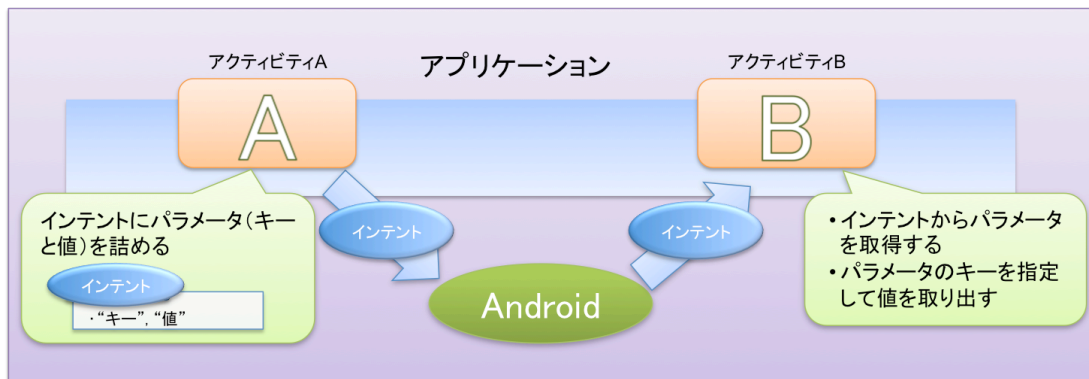
解答ドキュメントを参照して下さい。

6.5 画面遷移の連携

一覧画面から詳細画面に遷移する場合は、詳細画面は一覧画面で何が選択されたのかを知っておく必要があります。Intent を使ってデータを渡すことができます。

データの渡し方

Activity 間でデータを渡すには、画面遷移の際に作成する Intent オブジェクトに渡したいデータを詰め込みます。データはキー、バリューのペアで作成します。遷移先画面は、Activity のメソッドを使用して送付された Intent オブジェクトを取得し、キーを使用して、Intent オブジェクトからデータを取得します。



遷移元 Activity

遷移元 Activity では生成した Intent オブジェクトに `putExtra` メソッドで、データを追加します。データはキーとバリューのペアになります。ここでは、"message"というキーに"Android"という文字を追加しています。

リスト 6.11: 遷移元 Activity

```

1: public void onClickNextButton(View v){
2:     Intent intent = new Intent(this, NextActivity.class);
3:     intent.putExtra("message", "Android");
4:     startActivity(intent);
5: }

```

遷移先 Activity

遷移先の Activity で、`Activity#getIntent` メソッドを使用して、Intent オブジェクトを取得します。データの型に応じた `Intent#getExtra` メソッドを使用して、Intent オブジェクトに格納されているパラメータを取得する。ここでは"message"というキーを指定して"Android"という文字列を取得しています。

`getIntegerExtra:`

Integer 型

`getLongExtra:`

Long 型のデータを取得する

`getStringExtra:`

String 型のデータを取得する

リスト 6.14: 遷移先 Activity

```

1: protected void onCreate(Bundle savedInstanceState) {

```

```
2:         super.onCreate(savedInstanceState);
3:         setContentView(R.layout.activity_next);
4:
5:         Intent intent = getIntent();
6:         String message = intent.getStringExtra("message");
7:
8:     }
```

6.6 【実習】画面遷移 3



実習 3

前回の実習で作成したアプリケーションに、データの受け渡し機能を追加しましょう。MainActivity で EditText を用意し、入力された文字列を NextActivity で表示させます。

手順

1. MainActivity のレイアウトを修正する
2. MainActivity でデータを渡す処理を追加する
3. NextActivity でデータの受取処理を追加する

手順 1. MainActivity のレイアウトを修正する

MainActivity のレイアウトを次のように修正します。

リスト 6.13: activity_main.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <EditText
6:         android:id="@+id/edit_message"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content" >
9:     </EditText>
10:
11:     <Button
12:         ...
13:     />
14:
15: </LinearLayout>/**
16:
17: === 手順 2. MainActivity でデータを渡す処理を追加する
18: MainActivity で Intent にデータを追加します。
19: キー "message" を指定して、EditText に入力された文字列を設定します。EditText の入力文字は
    getText メソッドの戻り値に toString メソッドを使うと取得することが出来ます。
20:
21: //listnum[getIntent][遷移先 Activity]{
22: editText.getText().toString()
```

リスト 6.15: onClickNextButton メソッド

```
1:     public void onClickNextButton(View v){
2:         EditText editText = (EditText)findViewById(R.id.edit_message);
3:         Intent intent = new Intent(this, NextActivity.class);
4:         intent.putExtra("message", editText.getText().toString());
5:         startActivity(intent);
6:     }
```

手順 3. NextActivity でデータの受取処理を追加する

NextActivity でデータを受け取り、TextView に表示させます。Intent を取得し、キー "message" に格納されたデータを取得します。今回は文字列が格納されているので、getStringExtra メソッドを使います。

リスト 6.16: onCreate メソッド

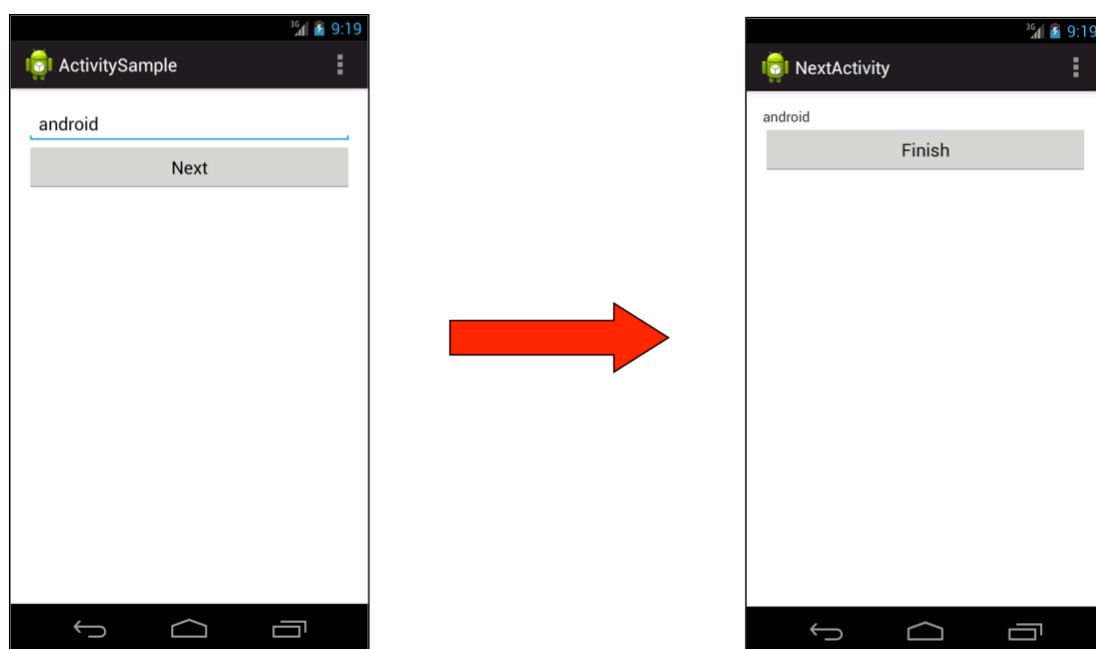
```
1:     protected void onCreate(Bundle savedInstanceState) {
2:         super.onCreate(savedInstanceState);
3:         setContentView(R.layout.activity_next);
4:
5:         Intent intent = getIntent();
6:         String message = intent.getStringExtra("message");
```



```
7:     TextView textView = (TextView)findViewById(R.id.text_message);
8:     textView.setText(message);
9:
10: }
```

確認

アプリケーションを実行し、EditText に入力した文字列が NextActivity で表示させるところを確認します。



解答

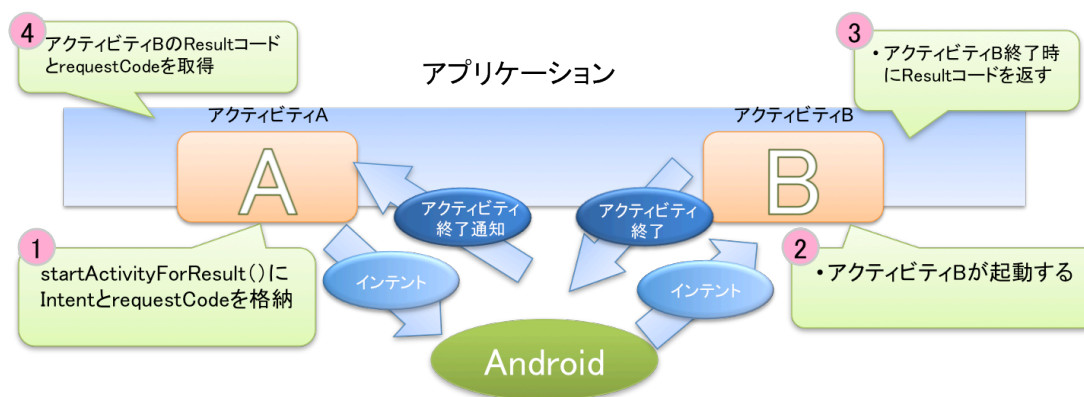
解答ドキュメントを参照してください。

6.7 遷移先画面から結果を受け取る

遷移先の Activity と連携して、終了時に値を受け取ることができます。遷移先画面で OK が選択されたのか、Cancel が選択されたのかなどに利用することができます。

結果を受け取る方法

遷移元の画面で `startActivityForResult` メソッドで Activity を起動します。このメソッドで起動した Activity を終了すると、呼び出し元の Activity の `onActivityResult` がコールバックされます。引数には終了コードやデータなどが渡され、Activity がどのように終了したのかを判定することができます。



Activity の処理結果を取得する（遷移元 Activity）

遷移元 Activity では、呼び出し処理と、遷移先 Activity 終了時のコールバック処理を追加します

画面を呼び出す処理 画面を呼び出すときは、`startActivityForResult` を使います。第 1 引数の Intent へ起動したい Activity をセットします。第 2 引数には起動 Activity の識別コードを指定します。

リスト 6.17: `startActivityForResult`

```
1: public void startActivityForResult (Intent intent, int requestCode)
```

遷移先 Activity 終了時のコールバック処理 遷移先 Activity が終了すると、`onActivityResult` がコールバックされます。このメソッドをオーバーライドし、遷移先 Activity の終了処理に応じた処理を追加します。第 1 引数には、終了した Activity のリクエストコードが各のされています。この引数を使って、どの Activity からの終了なのかを判定します。第 2 引数には、Activity がどのように終了したのかを表すコードが格納されています。第 3 引数には Intent が格納されています。終了

コード以外の情報やデータは Intent から取り出します。

リスト 6.18: onActivityResult

```
1: protected void onActivityResult (int requestCode, int resultCode, Intent data)
```

リスト 6.19: 遷移元 Activity の例

```
1: public void onClickNextButton(View v){
2:     EditText editText = (EditText)findViewById(R.id.edit_message);
3:     Intent intent = new Intent(this, NextActivity.class);
4:     intent.putExtra("message", editText.getText().toString());
5:     startActivityForResult(intent, 123);
6: }
7:
8: @Override
9: protected void onActivityResult(int requestCode, int resultCode, Intent data) {
10:     if(requestCode == 123){
11:         Log.v("MainActivity",
12:             "NextActivity が終了しました。終了コード=" + resultCode);
13:     }
14: }
```

Activity の処理結果を取得する (遷移先 Activity)

遷移先の Activity では、setResult メソッドを使って、どのように終了したのかを設定します。第 1 引数には終了コードを設定します。第 2 引数には終了コード以外のデータを設定します。第 2 引数は省略することができます。終了コードには、一般的には RESULT_CANCELED、RESULT_OK が用いられるが任意のコードを返すことも可能

リスト 6.20: 遷移先 Activity の例

```
1: public static final int RESULT_CANCELED = 0
2: public static final int RESULT_OK = -1
```

リスト 6.21: 遷移先 Activity の例

```
1: public void onClickFinishButton(View v){
2:     setResult(RESULT_OK);
3:     finish();
4: }
```

requestCode と resultCode

起動した Activity がどのように終了したのかを識別するためのコードが、requestCode と resultCode です。

requestCode:

起動した Activity からの実行結果を受け取ることで、どの Activity からの起動なのかを識別するコード

resultCode:

処理の結果コード int 型の値が戻り値となる。実行結果によって返す結果コードを変化させるなどして使用する。

6.8 【実習】画面遷移 4



実習 4

実習 3 で作成したアプリケーションに終了結果の受け取り処理を追加します。

手順

1. MainActivity で終了結果を受け取る処理を追加する
2. NextActivity で終了結果を渡す処理を追加する

手順 1. MainActivity で終了結果を受け取る処理を追加する

startActivityForResult メソッドを使って、リクエストコード"123"を指定して NextActivity を起動します。NextActivity 終了時の処理を追加します。

リスト 6.22: MainActivity

```
1: public void onClickNextButton(View v){
2:     EditText editText = (EditText)findViewById(R.id.edit_message);
```

```
3:     Intent intent = new Intent(this, NextActivity.class);
4:     intent.putExtra("message", editText.getText().toString());
5:     startActivityForResult(intent, 123);
6: }
7:
8: @Override
9: protected void onActivityResult(int requestCode, int resultCode, Intent data) {
10:     if(requestCode == 123){
11:         Log.v("MainACTivity",
12:             "NextActivity が終了しました。終了コード=" + resultCode);
13:     }
14: }
```

手順 2. NextActivity で終了結果を渡す処理を追加する

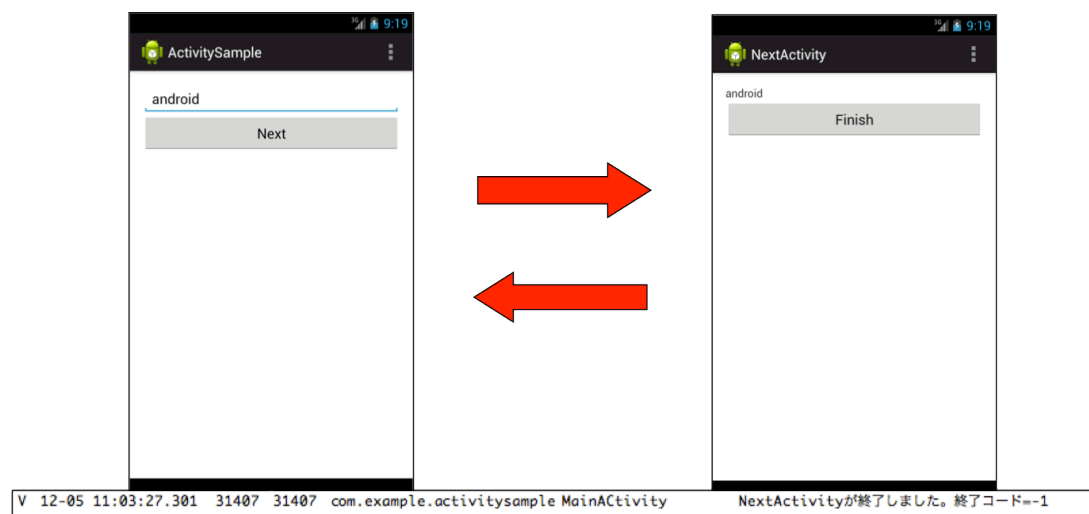
setResult メソッドを使って、終了結果をセットします。終了コードには RESULT_OK を指定します。

リスト 6.23: NextAcitvity

```
1:     public void onClickFinishButton(View v){
2:         setResult(RESULT_OK);
3:         finish();
4:     }
```

確認

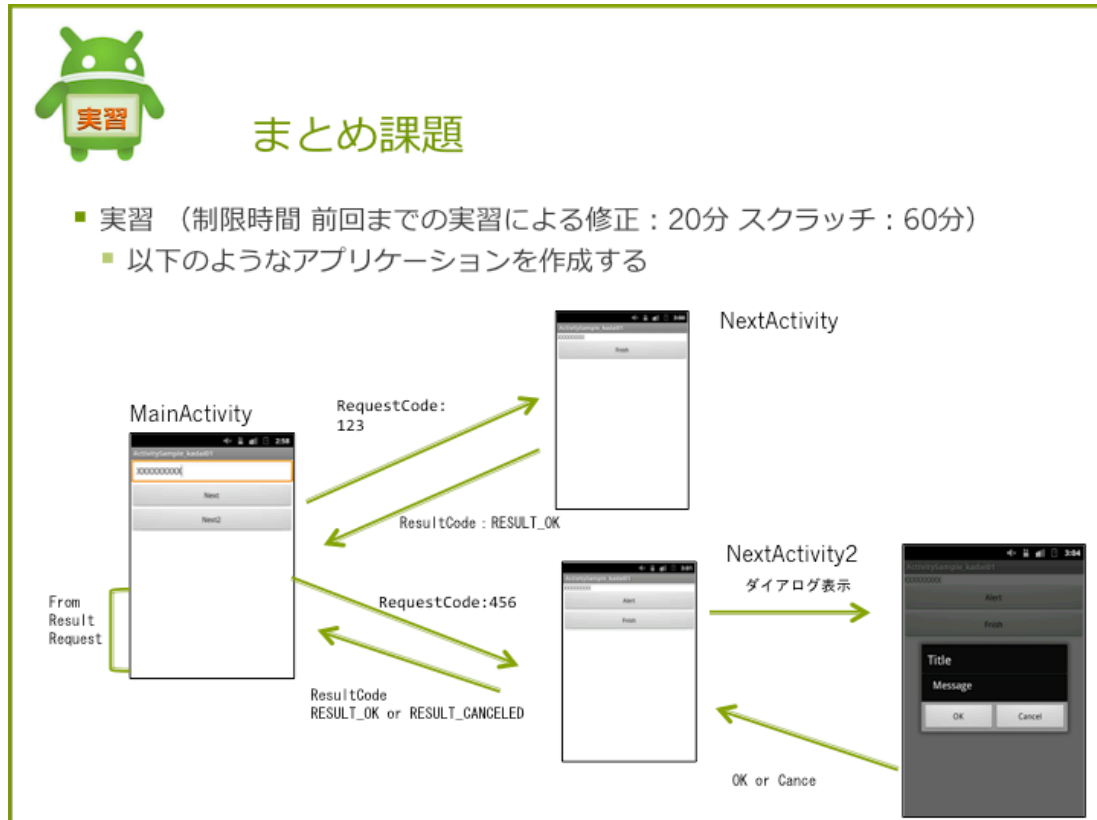
アプリケーションを実行し、NextActivity 終了時に、「NextActivity が終了しました。終了コード=-1」というログが出力されることを確認します。



解答

解答ドキュメントを参照してください。

6.9 まとめ課題



MainActivity, NextActivity, NextActivity2 の3つの画面からなるアプリケーションを作成しましょう。

アプリケーション動作概要

MainActivity は、NextActivity または、NextActivity2 を起動します。MainActivity は、それぞれの Activity が終了したとき起動した Activity に対応したログを出力します。

1. 「Next」 or 「Next2」 ボタンクリック時に RequestCode を指定して Activity を起動する

RequestCode には、以下の値を設定します。

- NextActivity :123
- NextActivity2 :456

2. 起動した Activity は ResultCode を指定して終了する

ResultCode には、以下の値を設定します。

- NextActivity : RESULT_OK
- NextActivity2 : AlertDialog で選択したボタンで以下の値を返す
 - 「 OK 」 を選択 : RESULT_OK
 - 「 Cancel 」 を選択 : RESULT_CANCELED

3. 起動元 Activity に終了結果の情報を表示する

各 TextView には以下の情報を表示させます。

- From : 起動した Activity 名
- Result : ResultCode
- Request : RequestCode



図: 動作例

解答

解答ドキュメントを参照してください。

第7章

ユーザインタフェース 2

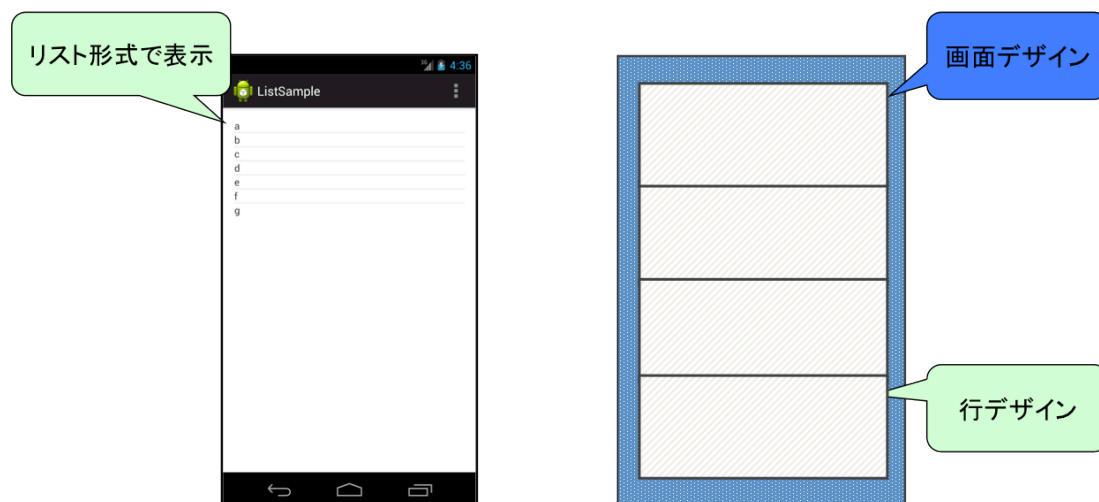
この章の目的

- Adapter について理解する
- ListView の使い方を理解する
- Spinner の使い方を理解する

7.1 ListView

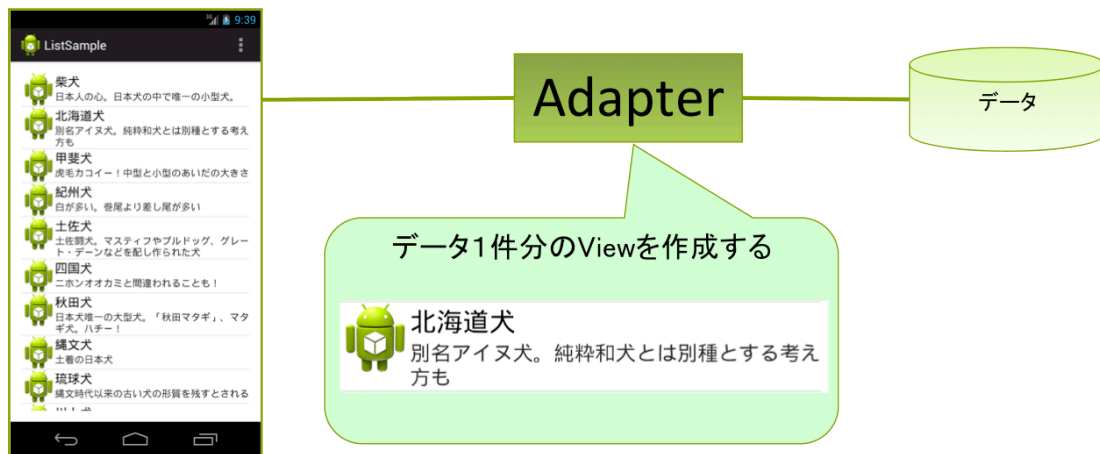
ListView とは

ListView はデータをリスト形式で表示するためのビューグループです。ListView を使用するときには画面デザインと行デザインの2つのリソースファイルが必要です。行のデザインと画面デザインとの関連付けは、Adapter クラスを使ってプログラムで制御します。



Adapter とは

ListView などのデータを、一覧形式で表示する ViewGroup のインタフェースとして Adapter クラスが用意されています。Adapter がデータ1件分の View を、組み立てる役割を担当しています。



View の組み立ての仕組み

Adapter はデータとビューの連携を担っており、表示すべきビューをデータから組み立てるのが役割です。データ 1 件分をどのような View 構造で表示するかを決定するのが `getView` メソッドです。

getView:

`getView` メソッドは、新しいデータが表示されるタイミングで呼び出されます。ListView の場合、画面をスクロールして、画面外から新しいデータが表示されるタイミングで呼び出されます。

ArrayAdapter

Android にはデータの管理方法に適した Adapter クラスが複数用意されています。ArrayAdapter クラスは配列や List などのデータ形式を管理するのに適した Adapter です。コンストラクタを使ってオブジェクトを生成します。第 1 引数に Context、第 2 引数に行のリソース ID を指定します。第 3 引数には管理するデータ型の配列またはコレクションを指定します。

```
ArrayAdapter(Context context, int resource, T[] objects)
```

```
ArrayAdapter(Context context, int resource, List<T> objects)
```

ListView の `setAdapter` メソッドまたは、ListActivity の `setListAdapter` を使って ListView との関連付けを行います。

```
void setAdapter(ListAdapter adapter)
```

```
void setListAdapter(ListAdapter adapter)
```

ListView の使い方

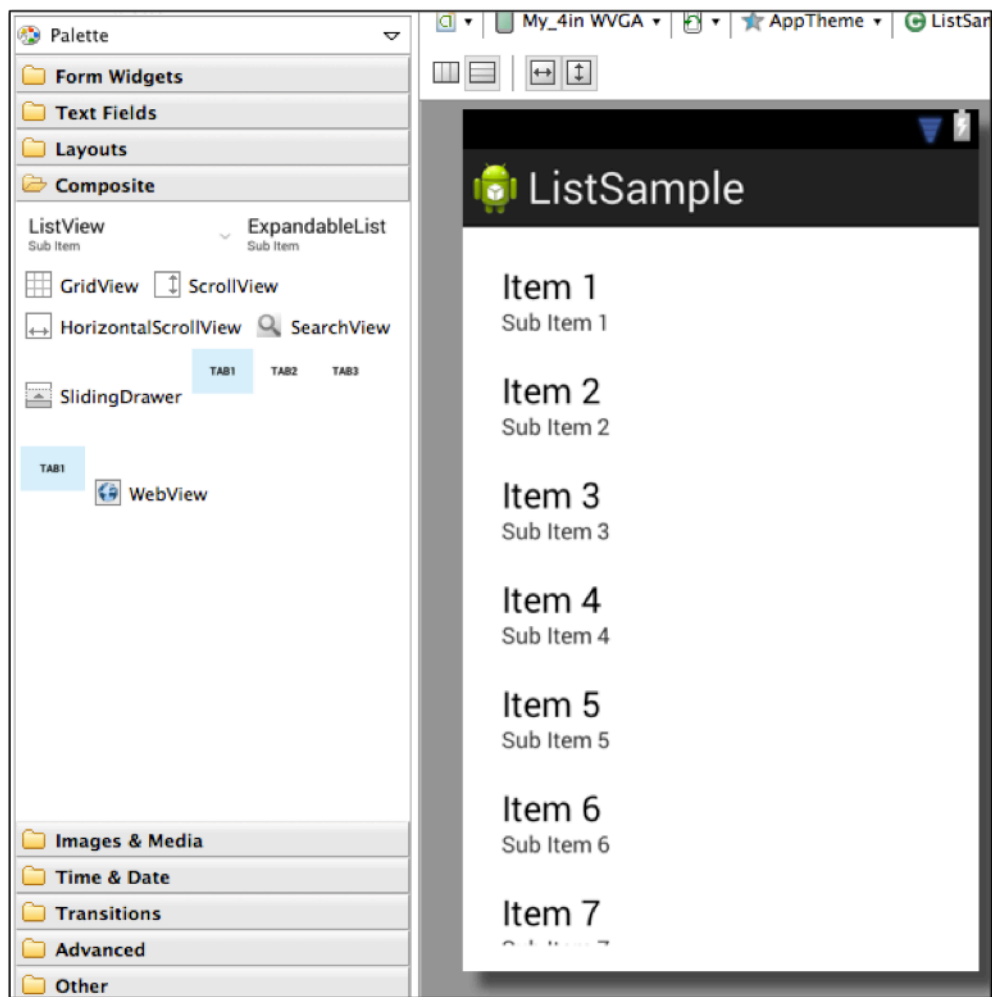
ListView と ListActivity を使って、シンプルな一覧画面を作ってみましょう。

手順

1. レイアウトファイルに ListView を追加する
2. 行のレイアウトファイルを作成する
3. Activity を修正する

手順 1. レイアウトファイルに ListView を追加する

レイアウトファイルをエディタで開き、ListView を画面上にドラッグ&ドロップします。



プロパティの変更

ListActivity を使って ListView にデータを表示する場合は、id に "@android:id/list" を指定する必要があります。

id:

@android:id/list

リスト 7.1: ListView の例

```
1: <ListView
2:     android:id="@android:id/list"
3:     android:layout_width="match_parent"
4:     android:layout_height="wrap_content" >
5: </ListView>
```

手順 2. 行のレイアウトファイルを作成する

re/layout フォルダを選択し、AndroidXML 作成ボタンをクリックします。File 名に「list_row.xml」と指定し、「Finish」ボタンをクリックし、Window を閉じます。

行のレイアウトファイルを修正する

ArrayAdapter を使って一覧を表示する場合は、行のレイアウトファイルは TextView だけで構成されている必要があります。

リスト 7.2: list_row.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <TextView xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:id="@+id/textView1"
4:     android:layout_width="match_parent"
5:     android:layout_height="wrap_content" />
```

手順 3. Activity を修正する

Activity を修正し、ListView にデータを表示させます。

1. 継承元を ListActivity に変更する

継承元を ListActivity に変更し、表示データの String 配列を定義します。

リスト 7.3: 継承元を ListActivity に変更する

```
1: import android.app.ListActivity;
2:
3: public class ListSampleActivity extends ListActivity {
4:
5:     private static final String[] ITEMS = {"a", "b", "c", "d", "e", "f", "g"};
```

2. ArrayAdapter を生成する

コンストラクタを使って ArrayAdapter を生成します。初期化と同時にデータの追加を行います。

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.list_row, ITEMS);
```

3. ListView に Adapter をセットする

```
setListAdapter(adapter);
```

全体ソース

Activity の修正をまとめると、以下のようになります。

リスト 7.4: ListView のサンプル

```
1: public class ListSampleActivity extends ListActivity {
2:
3:     private static final String[] ITEMS = {"a", "b", "c", "d", "e", "f", "g"};
4:
5:     @Override
6:     protected void onCreate(Bundle savedInstanceState) {
7:         super.onCreate(savedInstanceState);
8:         setContentView(R.layout.activity_list_sample);
9:
10:        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
11:            this, R.layout.list_row, ITEMS);
12:        setListAdapter(adapter);
13:    }
14:
15:    @Override
16:    public boolean onCreateOptionsMenu(Menu menu) {
17:        // Inflate the menu; this adds items to the action bar if it is present.
18:        getMenuInflater().inflate(R.menu.list_sample, menu);
19:        return true;
20:    }
21:
22: }
```

7.2 【実習】ListView 1



- 実習 1
 - ListViewを使ったサンプルアプリケーションを作成する

実習 1 ListView の表示

シンプルな ListView を使ったアプリケーションを作成しましょう。

プロジェクト概要

表 7.1 プロジェクト概要

項目	設定値
Project Name	ListSample
Build Target	4.4
Aplication name	ListSample
Package	com.example.listsample
Create Activity	ListSampleActivity
Layout File	activity_list_sample

手順

1. レイアウトファイルを修正する
2. 行用のレイアウトファイルを作成する
3. Activity に ListView の表示処理を追加する

手順 1. レイアウトファイルを修正する

ListView を定義した画面デザインのレイアウトファイル"activity_list_sample.xml"に ListView を追加します。

特別な設定が必要なプロパティ

ListView の id の値に中止してください。

id:

@android:id/list

リスト 7.5: activity_list_sample.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <ListView
6:         android:id="@android:id/list"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content" >
9:     </ListView>
10:
11: </LinearLayout>
```

手順 2. 行用のレイアウトファイルを作成する

ListView 内で使用する、行デザインのレイアウトファイル"list_row.xml"を作成する。

リスト 7.6: list_row.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <TextView xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:id="@+id/textView1"
4:     android:layout_width="match_parent"
5:     android:layout_height="wrap_content" />
```

手順 3. Activity に ListView の表示処理を追加する

継承元を ListActivitiy に変更し、Adapter を使用して ListView にデータを表示されます。

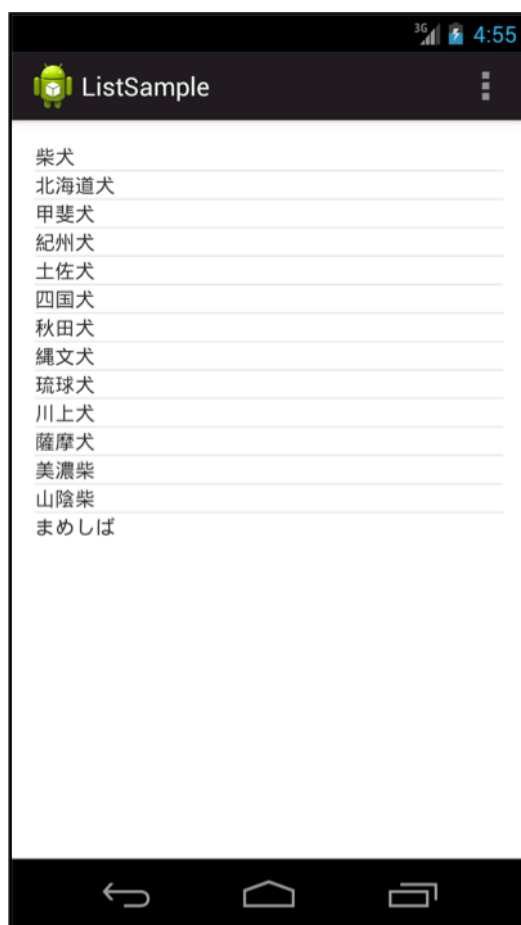
1. 継承元を ListActivity に変更する
2. ArrayAdapter を生成する
3. ListView に Adapter をセットする

リスト 7.7: ListSampleActivity.java

```
1: package com.example.listsample;
2:
3: import android.os.Bundle;
4: import android.view.Menu;
5: import android.widget.ArrayAdapter;
6: import android.app.ListActivity;
7:
8: public class ListSampleActivity extends ListActivity {
9:
10:     private static final String[] ITEMS = { "柴犬", "北海道犬", "甲斐犬",
11:         "紀州犬", "土佐犬", "四国犬", "秋田犬", "縄文犬", "琉球犬", "川上犬",
12:         "薩摩犬", "美濃柴", "山陰柴", "まめしば" };
13:
14:     @Override
15:     protected void onCreate(Bundle savedInstanceState) {
16:         super.onCreate(savedInstanceState);
17:         setContentView(R.layout.activity_list_sample);
18:
19:         ArrayAdapter<String> adapter = new ArrayAdapter<String>(
20:             this, R.layout.list_row, ITEMS);
21:         setListAdapter(adapter);
22:     }
23:
24:     @Override
25:     public boolean onCreateOptionsMenu(Menu menu) {
26:         // Inflate the menu; this adds items to the action bar if it is present.
27:         getMenuInflater().inflate(R.menu.list_sample, menu);
28:         return true;
29:     }
30:
31: }
```

実行結果

アプリケーションを実行し、一覧が表示されることを確認します。



解答

解答ドキュメントを参照してください。

7.3 一覧のアイテムを選択する

一覧に表示されているアイテムをクリックしても何も起きません。アイテムの選択イベントを受け取れるようにしましょう。

選択イベントを取得する


ListView のアイテム選択を認識するには、ListActivity#onListItemClick メソッドを実装します。第1引数に選択した ListView、第2引数に選択した行の View、第3引数に一覧の位置、第4引数にアイテム ID が渡されます。アイテム ID は使用する Adapter によって返す値が違います。ArrayAdapter の場合は要素番号が渡されます。

```
protected void onListItemClick(ListView l, View v, int position, long id)
```

リスト 7.8: onListItemClick のサンプル


```
1: public class ListSampleActivity extends ListActivity {  
2:     ... 略...  
3:     @Override  
4:     protected void onListItemClick(ListView l, View v, int position, long id){  
5:         Log.v("ListSample", "position = " + position);  
6:     }  
7: }
```

7.4 【実習】ListView 2



ListView 2

- 実習 2
 - 前項で作成したプログラムを修正し、一覧形式で表示したデータを選択すると、ログが出力される



Len	Time	PID	TID	Application	Tag	Text
D	12-05 06:31:46.831	13894	13898	com.example.listsample	dalvikvm	GC_CONCURRENT freed <1K, 5% free 3761K/3
D	12-05 06:31:46.401	13894	13894	com.example.listsample	galloc_goldfish	Emulator without GPU emulation detected.
V	12-05 06:31:48.101	13894	13894	com.example.listsample	ListSample	position = 2
V	12-05 06:31:52.071	13894	13894	com.example.listsample	ListSample	position = 5
V	12-05 06:31:53.271	13894	13894	com.example.listsample	ListSample	position = 9
V	12-05 06:31:54.711	13894	13894	com.example.listsample	ListSample	position = 11
V	12-05 06:31:55.781	13894	13894	com.example.listsample	ListSample	position = 12

実習 2 アイテム選択

実習 1 で作成したアプリケーションを修正し、アイテム選択時にログを出力させてみましょう。

ログ内容

```
Log.v("ListSample", "position = " + position);
```

確認

アプリケーションを実行し、アイテム選択時にログが出力されることを確認します。



Level	Time	PID	TID	Application	Tag	Text
D	12-05 06:31:46.031	13894	13898	com.example.listsample	dalvikvm	GC_CONCURRENT freed <1K, 5% free 3761K/3
D	12-05 06:31:46.401	13894	13894	com.example.listsample	gralloc_goldfish	Emulator without GPU emulation detected.
V	12-05 06:31:48.101	13894	13894	com.example.listsample	ListSample	position = 2
V	12-05 06:31:52.071	13894	13894	com.example.listsample	ListSample	position = 5
V	12-05 06:31:53.271	13894	13894	com.example.listsample	ListSample	position = 9
V	12-05 06:31:54.711	13894	13894	com.example.listsample	ListSample	position = 11
V	12-05 06:31:55.781	13894	13894	com.example.listsample	ListSample	position = 12

解答

解答ドキュメントを参照してください。

7.5 ListView のカスタマイズ

ListView をカスタマイズして、使いやすいユーザーインターフェースを作成することができます。



customlistview_example

ListView をカスタマイズする方法

1 行に、「サムネイル」「タイトル」「説明」の 3 つの要素を含んだ ListView を作成してみましょう。

手順

1. 行のレイアウトを修正する
2. データクラスを作成する
3. ArrayAdapter を継承した独自の Adapter を作成する
4. ListView にデータを表示させる

手順 1. 行のレイアウトを修正する

LinearLayout を組み合わせて図 7.1 のような配置で、ImageView、TextView、TextView を表示できるようにします。xml の内容はリスト 7.9 のようになります。

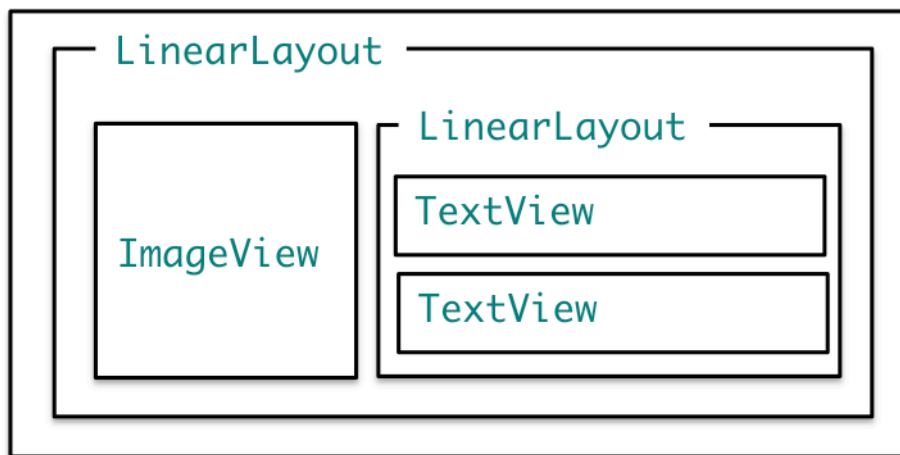


図 7.1 レイアウト構成

リスト 7.9: list_row.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     ...
4:     >
5:
6:     <ImageView
7:         android:id="@+id/image_thumbnail"
8:         android:layout_width="wrap_content"
9:         android:layout_height="wrap_content" >
10:    </ImageView>
11:
12:    <LinearLayout
13:        android:id="@+id/LinearLayout01"
14:        android:layout_width="wrap_content"
15:        android:layout_height="wrap_content"
16:        android:orientation="vertical" >
17:
18:        <TextView
19:            android:id="@+id/text_title"
20:            android:layout_width="wrap_content"
21:            android:layout_height="wrap_content"
22:            android:textAppearance="?android:attr/textAppearanceMedium" >
23:        </TextView>
24:
25:        <TextView
26:            android:id="@+id/text_detail"
27:            android:layout_width="wrap_content"
28:            android:layout_height="wrap_content"
29:            android:textAppearance="?android:attr/textAppearanceSmall" >
30:        </TextView>
31:    </LinearLayout>
32:
33: </LinearLayout>
```


手順 2. データクラスを作成する

1 行分の表示データ、「サブメールイメージ」「タイトル」「説明」をメンバ変数としたデータクラス「Item」を作成します。

リスト 7.10: Item.java

```
1: package com.example.listsample;
2:
3: public class Item {
4:     public String title;
5:     public String detail;
6:     public int resourceId;
7: }
```

手順 3. 独自の Adapter を作成する

Activity の内部クラスに、ArrayAdapter を継承した独自の Adapter を作成します。ジェネリックスには Item クラスを指定します。

リスト 7.11: ItemAdapter クラス

```
1:     class ItemAdapter extends ArrayAdapter<Item> {
2:
3:         public ItemAdapter(Context context) {
4:             super(context, R.layout.list_row);
5:         }
6:
7:         @Override
8:         public View getView(int position, View convertView, ViewGroup parent) {
9:             if (convertView == null) {
10:                 convertView = getLayoutInflater().inflate(R.layout.list_row, null);
11:             }
12:
13:             Item item = getItem(position);
14:             // TODO ImageView の設定
15:             ImageView imageView = (ImageView) convertView
16:                 .findViewById(R.id.image_thumbnail);
17:             imageView.setImageResource(item.resourceId);
18:
19:             // TODO TextView (Title) の設定
20:             TextView textTitle = (TextView) convertView
21:                 .findViewById(R.id.text_title);
22:             textTitle.setText(item.title);
23:
24:             // TODO TextView (summary) の設定
25:             TextView textSummary = (TextView) convertView
26:                 .findViewById(R.id.text_detail);
27:             textSummary.setText(item.detail);
28:             return convertView;

```

```
29:         }
30:
31:     }
```

手順 4. ListView にデータを表示させる

MainActivity の onCreate メソッドで setListAdapter で独自の Adapter をセットします。リソースファイルにタイトルと説明を配列で定義します。

string-array リソース 文字列の配列をリソースファイルに定義することができます。<string-array>タグを使って、リソース ID と配列の内容を定義します。プログラムからは、Resources クラスの getStringArray メソッドでリソース ID を指定して取得します。

リスト 7.12: リソースの取得

```
1:         String[] titles = getResources().getStringArray(R.array.titles);
2:         String[] detailes = getResources().getStringArray(R.array.detailes);
```

リスト 7.13: 構文

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <resources>
3:     <string-array name="titles">
4:         <item>柴犬</item>
5:         <item>北海道犬</item>
6:         <item>甲斐犬</item>
7:         ...
8:     </string-array>
9:     <string-array name="detailes">
10:        <item>日本人の心。日本犬の中で唯一の小型犬。</item>
11:        <item>別名アイヌ犬。純粋和犬とは別種とする考え方も</item>
12:        <item>虎毛カコイー！ 中型と小型のあいだの大きさ</item>
13:        ...
14:    </string-array>
15:
16: </resources>
```

取得した配列から表示データを組み立てます。for 文で繰り返し、Adapter の add メソッドを使ってデータを追加します。onCreate の処理はリスト 7.14 のようになります。

リスト 7.14: onCreate メソッド

```
1:     @Override
2:     protected void onCreate(Bundle savedInstanceState) {
3:         super.onCreate(savedInstanceState);
```

```
4:     setContentView(R.layout.activity_list_sample);
5:
6:     // TODO ItemAdapater を生成する
7:     ItemAdapter adapter = new ItemAdapter(this);
8:     setListAdapter(adapter);
9:
10:    // Item の作成
11:    // TODO リソースからテキスト配列を取得する
12:    // null の置き換え
13:    String[] titles = getResources().getStringArray(R.array.titles);
14:    String[] details = getResources().getStringArray(R.array.details);
15:
16:    for (int i = 0; i < titles.length; i++) {
17:        Item item = new Item();
18:        item.title = titles[i];
19:        item.detail = details[i];
20:        item.resourceId = R.drawable.ic_launcher;
21:
22:        // TODO Adapter にデータを追加する
23:        adapter.add(item);
24:    }
25: }
```

実行結果

アプリケーションを実行すると図 7. に表示されます。



7.6 【実習】ListView 3



実習 3 ListView のカスタマイズ

1 行に、「サムネイル」「タイトル」「説明」の 3 つの要素を含んだ ListView を作成してみましょう。最初は、スケルトンプロジェクト「ListSample_skeleton03」を解答ドキュメントよりダウンロード修正してください。

手順

1. 行のレイアウトを修正する
2. データクラスを作成する
3. ArrayAdapter を継承した独自の Adapter を作成する
4. ListView にデータを表示させる

スケルトンプロジェクトについて

スケルトンプロジェクトは、未実装の処理が含まれています。未実装の処理を修正し、アプリケーションを完成させましょう。

実装済

- lis2. データクラスを作成する
 - ファイル名「Item.java」で定義済です。

未実装

- 行のレイアウトを修正する
 - ファイル名「list_row.xml」で新規に作成します。
- ArrayAdapter を継承した独自の Adapter を作成する
 - getView メソッドの一部の処理が未実装になっています。
- ListView にデータを表示させる
 - onCreate メソッドの一部の処理が未実装になっています。

手順 1. 行のレイアウトを修正する

3つの要素をもった list_row.xml を作成します。リスト 7.9 と同じ内容になります。

手順 2. データクラスを作成する

スケルトンプロジェクトでは作成済です。。リスト 7.10 と同じ内容になります。

手順 3. ArrayAdapter を継承した独自の Adapter を作成する

ListSampleActivity クラスに ArrayAdapter を継承した ItemAdapter クラスが内部クラスで定義済です。getView メソッドが未完成なので、TODO を埋めていきます。リスト 7.11 と同じ内容になります。

手順 4. ListView にデータを表示させる

ListSampleActivity クラスの onCreate メソッドの TODO を埋めて、ListView の表示処理を実装させます。リスト 7.14 と同じ内容になります。

確認

アプリケーションを実行して、のように表示されることを確認します。確認できたら、同じプログラムを最初から作ってみましょう。



解答

解答ドキュメントを参照してください。

7.7 Spinner

Spinner とは

Spinner は、データを一覧形式で表示するためのビューグループです。Spinner を選択すると、ダイアログが起動し、一覧が表示されます。Spinner を使うときは、Spinner を使用するときは画面デザインと Spinner を閉じているときのデザインと、一覧表示の行デザインの 2 つのリソースファイルが必要です。Spinner もデータの表示には Adapter を使用します。



Spinner の使い方

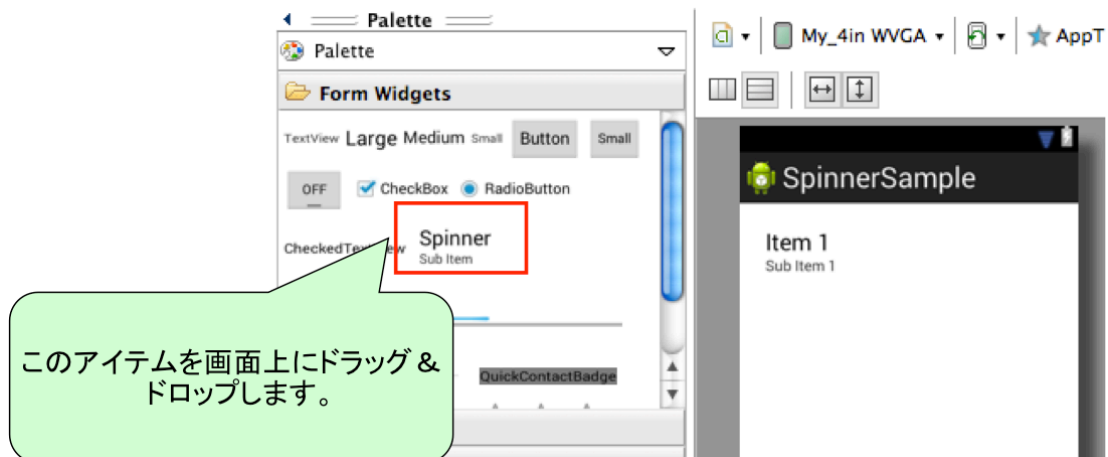
Spinner を使って、シンプルな一覧画面を作ってみましょう。

手順

1. レイアウトファイルに Spinner を追加する
2. Spinner にデータを追加する
3. Spinner が一覧表示しているときのレイアウトを設定する

手順 1. レイアウトファイルに ListView を追加する

レイアウトファイルをエディタで開き、Spinner を画面上にドラッグ&ドロップします。



リスト 7.15: Spinner の例

```

1: <Spinner
2:     android:id="@+id/spinner1"
3:     android:layout_width="match_parent"
4:     android:layout_height="wrap_content" />

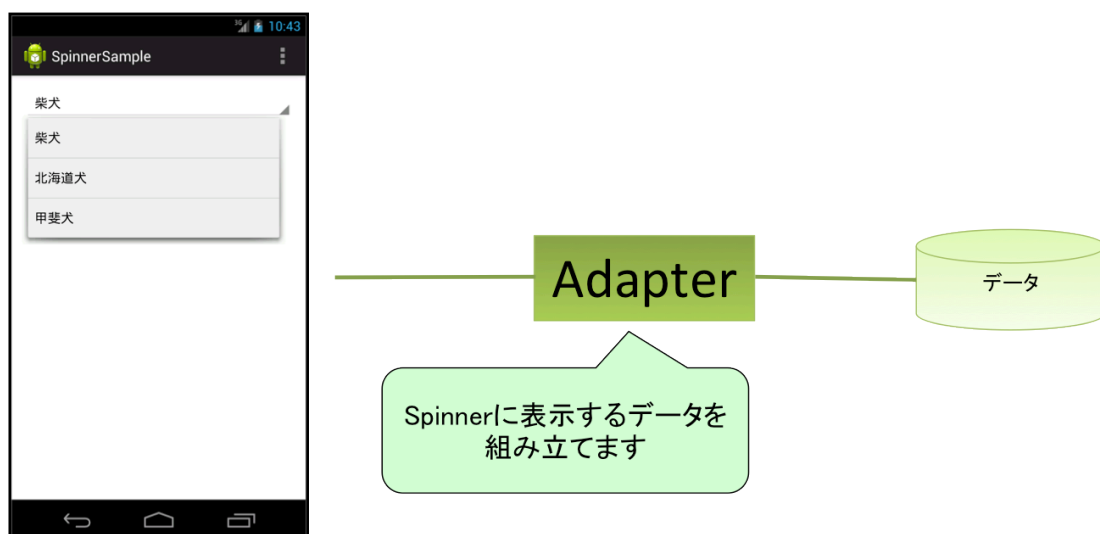
```

手順 2. Spinner にデータを追加する

Activity を修正し、Spinner が選択されたときのデザインと表示データの設定をします。

Adapter の準備

Spinner でもデータの表示に Adapter を使用します。ここでは ArrayAdapter を使います。



ArrayAdapter を使用して Spinner に表示するデータを追加するには2通りの方法があります。

- ArrayAdapter のインスタンスを生成後にデータを追加する方法
- ArrayAdapter 生成時にデータ配列を使用して作成する方法

ArrayAdapter のインスタンスを生成後にデータを追加する方法 ArrayAdapter クラスのコンストラクタを使用して ArrayAdapter のインスタンスを生成します。インスタンス生成後に add メソッドを使用して Spinner の一覧に表示するデータを追加します。第1引数に Context、第2引数に Spinner が閉じているときのレイアウトリソースを指定します。ここでは Android フレームワークで定義されている "simple_spinner_item" を指定します。Android フレームワークのリソース ID を使う場合は

```
android.R.layout.simple_spinner_item
```

のように指定します。

リスト 7.16: ArrayAdapter のインスタンスを生成後にデータを追加する

```
1: //ArrayAdapter インスタンスの生成
2: ArrayAdapter<String> adapter = new ArrayAdapter<String>
3:     (this, android.R.layout.simple_spinner_item);
4:
5: //ArrayAdapter のインスタンスに文字列を追加
6: adapter.add("柴犬");
7: adapter.add("北海道犬");
8: adapter.add("甲斐犬");
```

ArrayAdapter 生成時にデータ配列を使用して作成する方法 ArrayAdapter#createFromResource メソッドの第二引数に配列を指定して ArrayAdapter のインスタンスを生成します。配列には string-array リソースを指定します。ここではリスト 7.17 のリソースを使います。

リスト 7.17: 文字列リソース : dogs

```
1: <string-array name="dogs">
2:     <item>柴犬</item>
3:     <item>北海道犬</item>
4:     <item>甲斐犬</item>
5: </string-array>
```

リスト 7.18: ArrayAdapter 生成時にデータ配列を使用して作成する方法

```
1: adapter.createFromResource(this, R.array.dogs, android.R.layout.simple_spinner_item);
```

手順 3. Spinner が一覧表示しているときのレイアウトを設定する

Spinner の一覧に表示されるレイアウトの設定するには、`ArrayAdapter#setDropDownViewResource` メソッドを使用します。引数に表示されたときのリソース ID を指定します。ここでは Android フレームワークで定義されている `"simple_spinner_dropdown_item"` を指定します。

リスト 7.19: Spinner が一覧表示しているときのレイアウトを設定する

```
1: adapter.createFromResource(this, R.array.dogs, android.R.layout.simple_spinner_item);
```

全体ソース

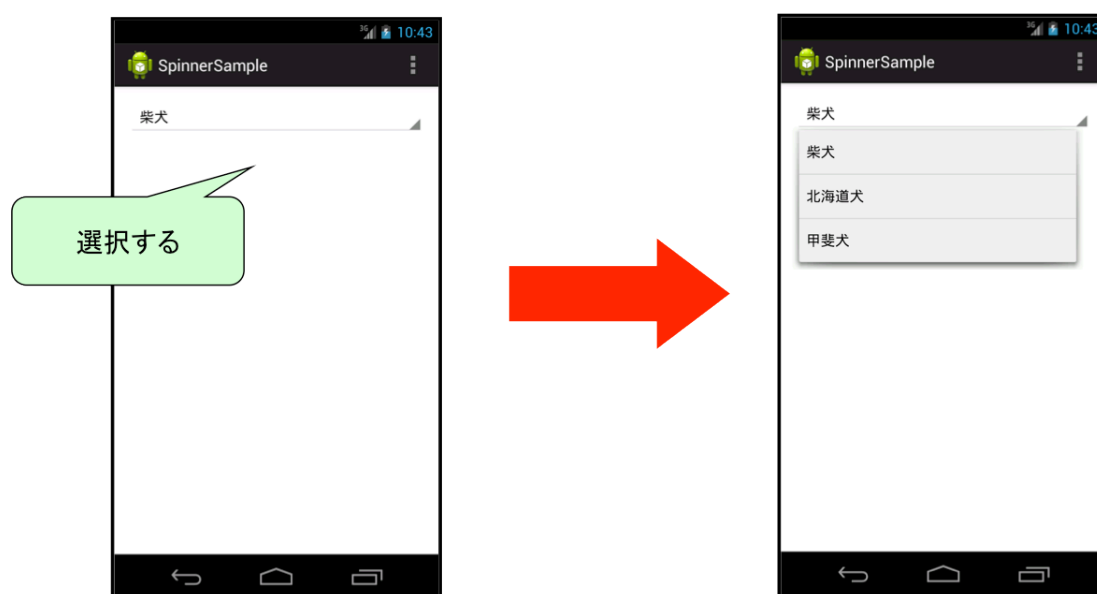
リスト 7.20: SpinnerSampleActivity.java

```
1: package com.example.spinnersample;
2:
3: import android.app.Activity;
4: import android.os.Bundle;
5: import android.view.Menu;
6: import android.widget.ArrayAdapter;
7: import android.widget.Spinner;
8:
9: public class SpinnerSampleActivity extends Activity {
10:
11:     @Override
12:     protected void onCreate(Bundle savedInstanceState) {
13:         super.onCreate(savedInstanceState);
14:         setContentView(R.layout.activity_spinner_sample);
15:
16:         // ArrayAdapter インスタンスの生成
17:
18:         ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this, R.array.dogs,
19:             android.R.layout.simple_spinner_item);
20:
21:         // リストに表示するためのレイアウトリソースを設定
22:         adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
23:
24:         // スピナーの生成
25:         Spinner spinner = (Spinner) findViewById(R.id.spinner1);
26:         // スピナーにアダプター設定
27:         spinner.setAdapter(adapter);
28:     }
29:
```

```
30:     @Override
31:     public boolean onCreateOptionsMenu(Menu menu) {
32:         // Inflate the menu; this adds items to the action bar if it is present.
33:         getMenuInflater().inflate(R.menu.spinner_sample, menu);
34:         return true;
35:     }
36:
37: }
```

実行結果

アプリケーションを実行し、Spinner を選択すると一覧が表示されることを確認します。



7.8 【実習】Spinner 1



実習 1 Spinner の表示

Spinner を表示するアプリケーションを作成しましょう。

プロジェクト概要

表 7.2 プロジェクト概要

項目	設定値
Project Name	SpinnerSample
Build Target	4.4
Application name	SpinnerSample
Package	com.example.spinnersample
Create Activity	SpinnerSampleActivity
Layout File	activity_spinner_sample

手順

string-array を作成する Spinner のリストに表示する文字列配列を追加
レイアウトファイルの修正 Spinner を追加
Activity に Spinner 表示用の処理を実装する。string-array から文字列を取得し Adapter を生成
Spinner に Adapter を設定

手順 1. string-array を作成する

Spinner のリストに表示する文字列配列を追加します。

リスト 7.21: 文字列リソース : dogs

```
1: <string-array name="dogs">
2:   <item>柴犬</item>
3:   <item>北海道犬</item>
4:   <item>甲斐犬</item>
5: </string-array>
```

手順 2. レイアウトファイルを修正する

Activity のレイアウトをリスト 7.22 のようにします。

リスト 7.22: activity_spinner_sample.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:   ...
3:   >
4:
5:   <Spinner
6:     android:id="@+id/spinner1"
7:     android:layout_width="match_parent"
8:     android:layout_height="wrap_content" />
9:
10: </LinearLayout>
```

手順 3. Activity に Spinner 表示用の処理を実装する。

string-array から文字列を取得し Adapter を生成します。Spinner に Adapter を設定します。

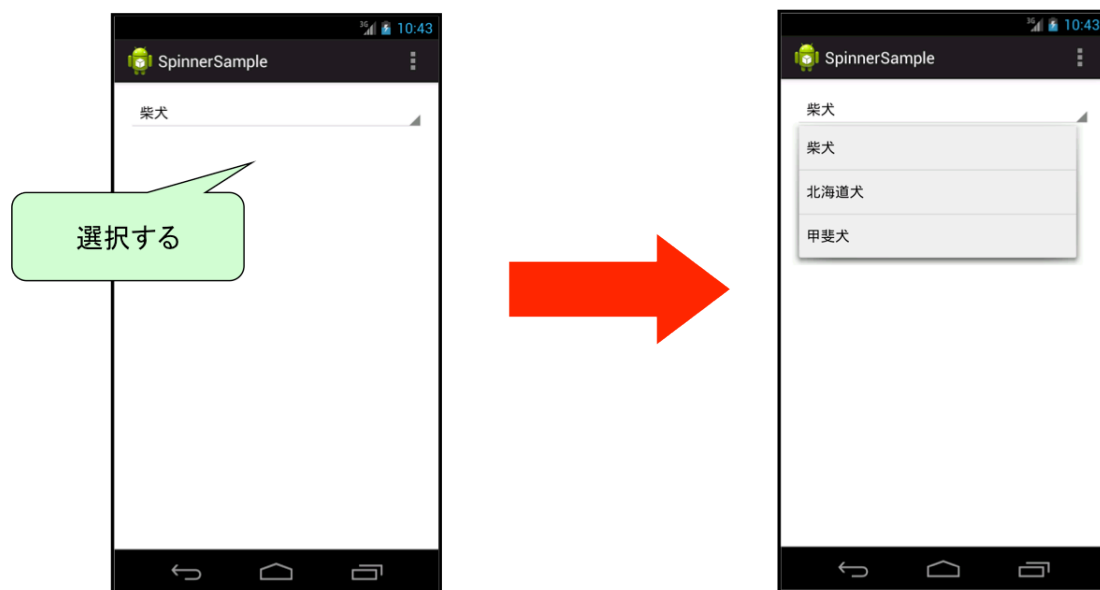
リスト 7.23: SpinnerSampleActivity

```
1: package com.example.spinnersample;
2:
```

```
3: import android.app.Activity;
4: import android.os.Bundle;
5: import android.view.Menu;
6: import android.widget.AdapterView;
7: import android.widget.Spinner;
8:
9: public class SpinnerSampleActivity extends Activity {
10:
11:     @Override
12:     protected void onCreate(Bundle savedInstanceState) {
13:         super.onCreate(savedInstanceState);
14:         setContentView(R.layout.activity_spinner_sample);
15:
16:         // ArrayAdapter インスタンスの生成
17:
18:         ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
19:             this, R.array.dogs, android.R.layout.simple_spinner_item);
20:
21:         // リストに表示するためのレイアウトリソースを設定
22:         adapter.setDropDownViewResource(
23:             android.R.layout.simple_spinner_dropdown_item);
24:
25:         // スピナーの生成
26:         Spinner spinner = (Spinner) findViewById(R.id.spinner1);
27:         // スピナーにアダプター設定
28:         spinner.setAdapter(adapter);
29:
30:     }
31:
32:     @Override
33:     public boolean onCreateOptionsMenu(Menu menu) {
34:         // Inflate the menu; this adds items to the action bar if it is present.
35:         getMenuInflater().inflate(R.menu.spinner_sample, menu);
36:         return true;
37:     }
38:
39: }
```

確認

アプリケーション実行し、Spinner が表示されることを確認します。



解答

解答ドキュメントを参照してください。

7.9 Spinner を選択する

一覧に表示されているアイテムをクリックしても何も起きません。アイテムの選択イベントを受け取れるようにしましょう。

選択イベントを取得する

Spinner では、`OnItemSelectedListener` を使って選択イベントを取得します。Spinner のリストから選択したアイテムを認識するには、`Spinner#setOnItemSelectedListener` を使って、`OnItemSelectedListener` を登録し、メソッドを実装します。

```
void setOnItemSelectedListener(AdapterView.OnItemSelectedListener listener)
```

`setOnItemSelectedListener` には、`onItemSelected` と `onNothingSelected` が定義されています。アイテムが選択されると、`onItemSelected` メソッドが呼び出されます。

```
void onItemSelected(AdapterView<?> parent, View view, int position, long id)
```

`onItemSelected` メソッドの第1引数に選択された Spinner オブジェクト、第2引数に選択した View、第3引数に位置、第4引数にアイテムの ID が入ります。ID は使用する Adapter によって様々ですが、`ArrayAdapter` の場合は要素番号が入ります。何も選択されなかった場合は `onNothingSelected` が呼び出されます。

リスト 7.24: `OnItemSelectedListener` の実装例

```
1: spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
2:     @Override
3:     public void onItemSelected(AdapterView<?> parent, View view,
4:         int position, long id) {
5:
6:         // リストを選んだ時の処理を記述
7:         TextView textView = (TextView) findViewById(R.id.textView1);
8:         textView.setText(parent.getSelectedItem().toString());
9:     }
10:
11:     @Override
12:     public void onNothingSelected(AdapterView<?> parent) {
13:         // 何も選択されなかった時の処理を記述
14:     }
15: });
```


選択したアイテムを取得する

`getSelectedItem` メソッドを使って Spinner で選択したアイテムを取得します。`getSelectedItem` メソッドは `Object` 型を返すため、管理しているデータ型に合わせてキャストする必要があります。

```
String item = (String) spinner.getSelectedItem();
```

`getSelectedItemPosition` メソッドを使うと選択されている位置を取得することができます。

```
int position = spinner.getSelectedItemPosition();
```

7.10 【実習】Spinner 2



実習 2 アイテム選択

実習 1 で作成したプログラムを修正し、Spinner で選択したデータを TextView に表示させてみましょう。

確認

Spinner で選択したデータを TextView に表示されていることを確認します。



解答

解答ドキュメントを参照してください。

第 8 章

HTTP 通信

この章の目的

- HTTP 通信について理解する
- WebAPI について理解する

8.1 Web サービスに接続する

ユーザはブラウザなどのクライアントを使って、HTTP 通信をして外部の Web サービスに接続します。Android は常時接続が可能のため、外部の Web サービスと連携したアプリケーションを作成することができます。特定の Web サービスに特化したアプリケーションを作成する場合は、プログラム上で通信ロジックを組み込む必要があります。



HTTP 通信の仕方

Android には HTTP 通信を行うユーティリティとして `DefaultHttpClient` クラスが用意されています。`DefaultHttpClient` クラスを使って HTTP 通信の仕方を見てください。

手順

1. `DefaultHttpClient` クラスを生成する。

2. リクエストメソッドの設定をする。
3. リクエストを発行しレスポンスオブジェクトを取得する
4. レスポンスステータスをチェックする。
5. レスポンスデータから必要な情報をとりだす。
6. AndroidManifest ファイルにインターネットアクセスのパーミッションを追加する
7. StrictMode の制限を解除する

手順 1. DefaultHttpClient クラスを生成する。

コンストラクタを使って DefaultHttpClient のオブジェクトを生成します。

```
// DefaultHttpClient オブジェクトの生成する
DefaultHttpClient client = new DefaultHttpClient();
```

手順 2. リクエストメソッドの設定をする

コンストラクタを使って、引数に接続先 URL を指定してリクエストオブジェクトを生成します。ここでは GET メソッドで接続する HttpGet オブジェクトを生成します。

```
// GET メソッドで接続するリクエストオブジェクトを生成する
HttpGet get = new HttpGet(URL);
```

手順 3. リクエストを発行しレスポンスオブジェクトを取得する

引数に生成したリクエストオブジェクトを指定し、execute メソッドでリクエストを発行します。execute メソッドを実行すると、HttpResponse オブジェクトが取得できます。

```
// リクエストを発行してレスポンスを取得する
HttpResponse res = client.execute(get);
```

手順 4. レスポンスステータスをチェックする。

取得した Response オブジェクトからステータスコードの確認をします。ステータスコードは HttpResponse クラスの getStatusLine の戻り値に対して getStatusCode メソッドを実行すると取得できます。HttpStatus インタフェースのステータスコード定数が用意されています。

```
// ステータスコードのチェックする
if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
}
```

表 8.1 主なステータスコード定数

定数	ステータスコード	説明
SC_OK	200	成功
SC_FORBIDDEN	403	アクセス禁止
SC_NOT_FOUND	404	接続先ページが見つからない
SC_INTERNAL_SERVER_ERROR	500	サーバサイドエラー

手順 5. レスポンスデータから必要な情報を取り出す

レスポンスデータから必要な情報を取得します。HttpEntity オブジェクトを取得し、必要に応じて対応するデータに変換します。(変換方法については後述)

```
HttpEntity entity = res.getEntity();
```

リスト 8.4: 手順 1 から 5 をまとめたコード

```
1: // DefaultHttpClient オブジェクトの生成する
2: DefaultHttpClient client = new DefaultHttpClient();
3:
4: // GET メソッドで接続するリクエストオブジェクトを生成する
5: HttpGet get = new HttpGet(URL);
6:
7: try {
8:     // リクエストを発行してレスポンスオブジェクトを取得する
9:     HttpResponse res = client.execute(get);
10:    // ステータスコードをチェックする
11:    if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
12:        // レスポンス情報を取得する
13:        HttpEntity entity = res.getEntity();
14:    }
15: } catch (Exception e) {
16:     Log.e(TAG, Log.getStackTraceString(e));
17: }
```

手順 6. AndroidManifest ファイルにインターネットアクセスのパーミッションを追加する

インターネット接続の権限を追加するため、AndroidManifest ファイルにパーミッションの設定をします。

<uses-permission>タグを追加し、 android:name 属性に"android.permission.INTERNET" を設定します。

```
HttpEntity entity = res.getEntity();
```

手順 7. StrictMode の制限を解除する

バージョン 2.3 以降より、StrictMode というものが追加されました。StrictMode はアプリケーションのメインスレッドでの実行制限です。バージョン 3.0 以降はデフォルトでメインスレッドでは Http 通信が行えないように設定されています。今回の例では、メインスレッドで HTTP 通信を行うため、以下のような方法で制限の解除を行います。

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_http_sample);  
  
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());  
}
```

8.2 【実習】HTTP 通信 1



実習 1 Http 通信をする

HTTP 通信を行いステータスコードをチェックするアプリケーションを作成しましょう。

通信概要

接続先:

(実習 1, 2) <http://www.oesf.jp/>

(実習 3) <http://iss.ndl.go.jp/api/openserch>

接続メソッド:

GET

ステータスコード:

200 (STATUS_OK)

プロジェクト概要

HttpSample_skeleton プロジェクトをインポートします。

表 8.2 プロジェクト概要

項目	設定値
Project Name	HttpSample
Build Target	4.4
Application name	HttpSample
Package	com.example.httpsample
Create Activity	HttpSampleActivity
Layout File	activity_http_sample

手順

スケルトンでは、いくつかの処理は実装済となっています。未実装の処理を実装し、プログラムを完成させましょう。

1. リソースファイルの修正 <実装済>
2. Activity に HTTP 通信処理の追加 <未実装>
3. AndroidManifest.xml の修正 <未実装>

手順 1. リソースファイルの修正 <実装済>

レイアウトファイルに Button を配置します。

リスト 8.2: activity_http_sample.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:     <Button
5:         android:id="@+id/button_connect_http"
6:         android:layout_width="match_parent"
7:         android:layout_height="wrap_content"
8:         android:onClick="onClickButton"
9:         android:text="@string/http" >
10:    </Button>
11:
```

手順 2. Activity に HTTP 通信処理の追加

Button が押されたときに HTTP 通信処理を追加します。Button が押されると、onClickButton メソッドが呼び出されます。onClickButton メソッドの 2 つの TODO に処理を追加します。

1. DefaultHttpClient クラスを生成する。

```
// TODO 【HTTP 通信 実習 1】No.01 DefaultHttpClient オブジェクトの生成する
DefaultHttpClient client = new DefaultHttpClient();
```

2. HttpGet オブジェクトを生成する。

コンストラクタを使って HttpGet オブジェクトを生成します。引数には定数 URL を使用します。

```
// TODO 【HTTP 通信 実習 1】No.02 GET メソッドで接続するリクエストオブジェクトを生成する
HttpGet get = new HttpGet(URL);
```

3. 指定した URL にリクエストを発行する。

execute メソッド実行し、対象接続先にリクエストを発行します。

```
// TODO 【HTTP 通信 実習 1】No.03 リクエストを発行してレスポンスを取得する
HttpResponse res = client.execute(get);
```

4. レスポンスステータスをチェックする。

ステータスが 200 ならログを出力します。
出力ログ : Log.v(TAG, "status ok");

```
if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
    // TODO 【HTTP 通信 実習 1】No.05 ログを出力する
    Log.v(TAG, "status ok");
    ...
}
```

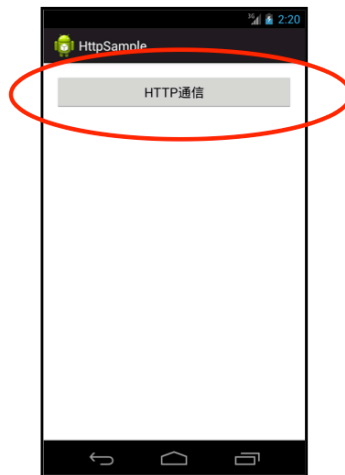
手順 3. AndroidManifest.xml の修正

インターネット接続のパーミッションを追加します

```
<!-- TODO インターネット接続のパーミッションを追加する -->
<uses-permission android:name="android.permission.INTERNET" >
</uses-permission>
```

確認

アプリケーションを実行し、接続成功のログが出力されることを確認します。



ログ

```
11-10 02:55... I 53  ActivityManager  Displayed activity jp.oesf.httpsample/.HttpSamp
11-10 02:55... I 53  ARMAssembler    generated scanline_00000077:03545404_00000A04
11-10 02:55... I 53  ARMAssembler    generated scanline_00000177:03515104_00001A01
11-10 02:56... D 230 dalvikvm       GC freed 3510 objects / 235960 bytes in 125ms
11-10 02:56... V 230  HttpClientSample status ok
```

解答

解答ドキュメントを参照してください。

8.3 レスポンスデータから必要な情報を取得する

取得したレスポンスデータから必要な情報を取り出します。レスポンスデータは HTTP ヘッダと HTTP ボディで構成されています。HTTP ボディには HTML 文や JSON など Web サーバから送られてくるデータが入っています。HttpResponse から HttpEntity オブジェクトを取得し、必要に応じて対応するデータ（文字列、画像、動画など）に変換します。

レスポンスデータから文字列を取得する方法

今回のケースではレスポンスデータは、HTML 文なので文字列に変換します。取得したレスポンスデータを、文字列に変換するには、HttpEntity クラスと EntityUtils を使用します。HttpEntity を文字列に変換するため、EntityUtils の toString メソッドを使用します。

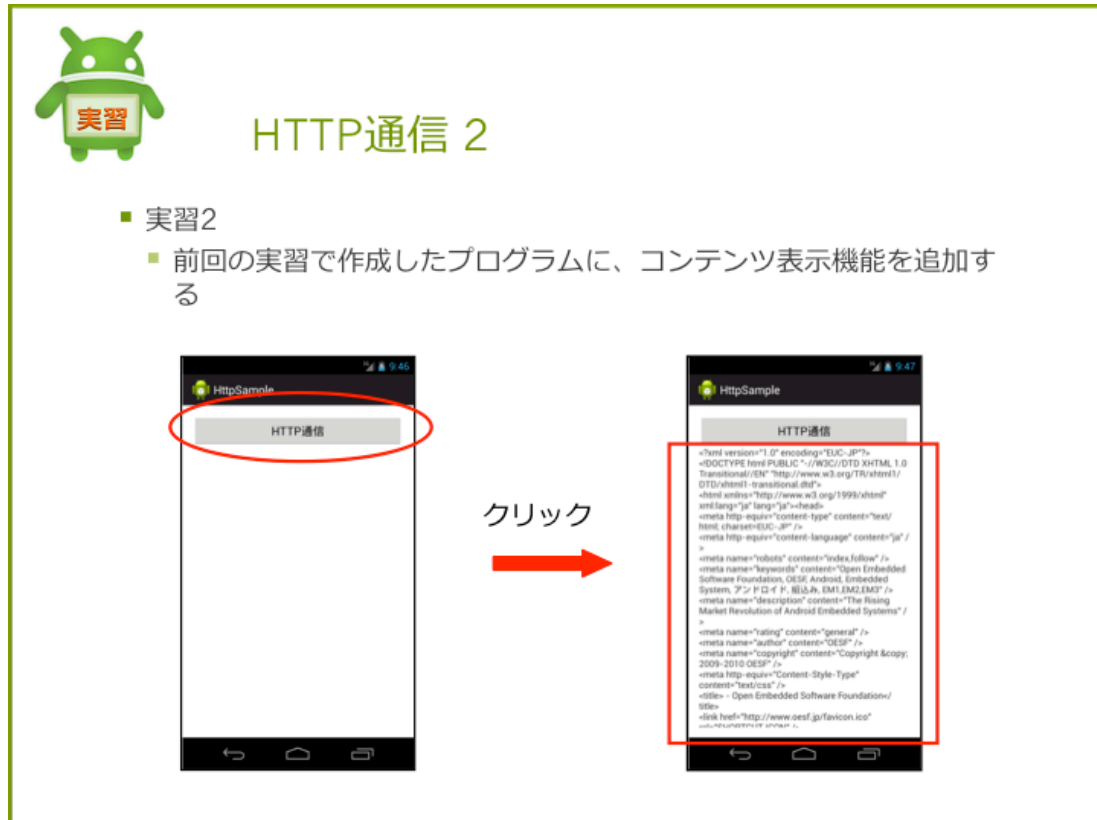
```
HttpResponse res = client.execute(get);
String content = EntityUtils.toString(res.getEntity(), "EUC-JP");
```

取得データの内容

今回の例では、OESF のホームページ (URL: <http://www.oesf.jp/index.html>) の HTML 文書が取得データになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dcndl="http://ndl.go.jp/dcndl/terms/" version="2.0"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:openSearch="http://a9.com/-/spec/opensearchrss/1.0/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:dcmitype="http://purl.org/dc/dcmitype/">
  <channel>
    <title>android - 国立国会図書館サーチ OpenSearch</title>
    <link>http://iss.ndl.go.jp/api/opensearch?title=android</link>
    <description>Search results for title=android </description>
    <language>ja</language>
    <openSearch:totalResults>1159</openSearch:totalResults>
    <openSearch:startIndex>1</openSearch:startIndex>
    <openSearch:itemsPerPage></openSearch:itemsPerPage>
    <item>
      <title>特集 199 ドルで破格の性能 驚異の Android タブレット 「Kindle Fire」徹底解剖</title>
      <link>http://iss.ndl.go.jp/books/R000000004-I023427272-00</link>
      ...
```

8.4 【実習】HTTP 通信 2



実習 2 レスポンスデータから文字列を取得する

実習 1 で作成したプログラムに、HTML 文を表示させましょう。

手順

1. レイアウトファイルの修正
2. Activity クラスにレスポンスデータ取得処理を追加
3. 取得したデータを画面に表示

手順 1. レイアウトファイルの修正

レイアウトファイルに、HTML 文を表示するための ScrollView と TextView を追加します。

リスト 8.3: activity_http_sample.xml

```
1: <!-- TODO 【HTTP 通信 実習 2】 TextView を追加する -->
2:
```

```
3:     <ScrollView
4:         android:id="@+id/scrollView1"
5:         android:layout_width="match_parent"
6:         android:layout_height="wrap_content" >
7:
8:         <LinearLayout
9:             android:layout_width="match_parent"
10:            android:layout_height="match_parent"
11:            android:orientation="vertical" >
12:
13:            <TextView
14:                android:id="@+id/text_content"
15:                android:layout_width="wrap_content"
16:                android:layout_height="wrap_content" >
17:            </TextView>
18:        </LinearLayout>
19:    </ScrollView>
```

手順 2. Activity クラスにレスポンスデータ取得処理を追加

HttpResponse から HttpEntity を取得し、EntityUtils#toString メソッドに HttpEntity を引数に指定して HTML 文を取得します。

```
// TODO 【HTTP 通信 実習 2】 No.01 html 文の取得
String content = EntityUtils.toString(res.getEntity(), "EUC-JP");
```

手順 3. 取得したデータを画面に表示

activity_http_sample.xml に追加した TextView にレスポンスデータを設定します。

```
// TODO 【HTTP 通信 実習 2】 No.02 TextView に取得したコンテンツデータを表示
TextView textView = (TextView)findViewById(R.id.textView1);
textView.setText(content);
```

確認

アプリケーションを実行し、HTML 文が画面に表示されることを確認します。



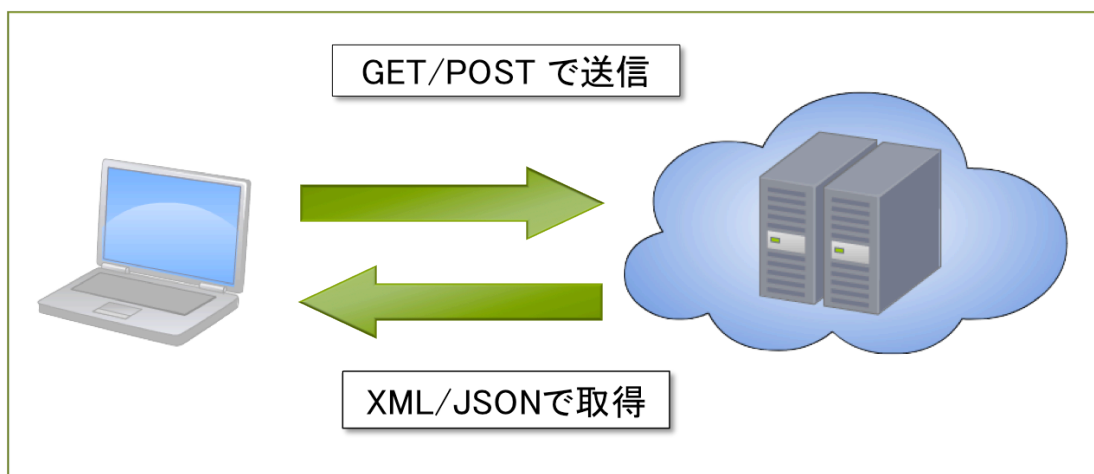
解答

解答別ドキュメントを参照してください。

8.5 WebAPI

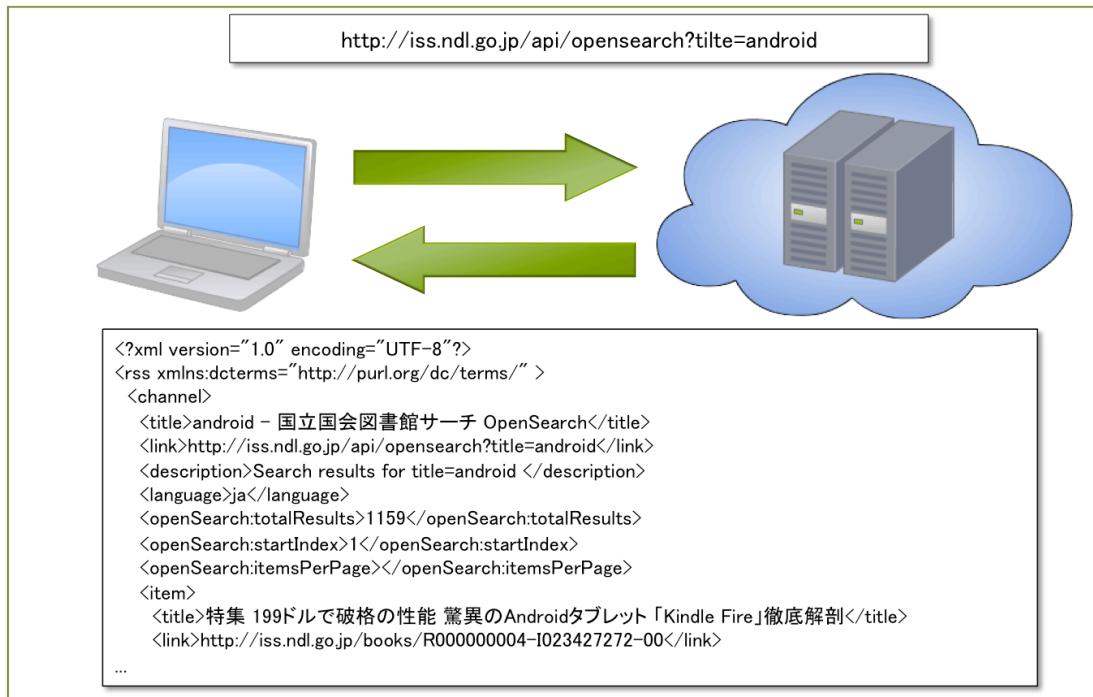
WebAPI とは

Web 上にあるアプリケーションプログラムインタフェースのことです。基本的には URL にパラメータを付与して、XML や JSON などのデータでやりとりを行います。代表的な物として、Yahoo API、Google API、Twitter API などがある。



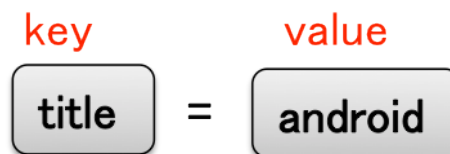
WebAPI を使った通信の仕方

WebAPI には、リクエストにパラメータを付与してアクセスします。リクエストは、各 Web API の URL(例 <http://iss.ndl.go.jp/api/opensearch>) に、パラメータ (例: `?title=android`) を追加して作成します。レスポンスは XML や JSON などで返されます。(上記の例では XML が返されます)



WebAPI パラメータ

パラメータはキー・バリューのペアで作成します。(例: `?title=android`)



パラメータは `key=value` の構成で、リクエスト URL の末尾に ? で接続します。

```
http://iss.ndl.go.jp/api/opensearch?title=android
```


WebAPI を使った HTTP 通信の実装方法

国会図書館 API を使って書籍を検索するアプリケーションを作成します。

手順

1. パラメータ付き URL の作成
2. HttpGet オブジェクトの作成
3. リクエストの送信

手順 1. パラメータ付き URL の作成

パラメータ付き URL の作成は、`android.net.Uri.Builder` クラスで作成します。

1. コンストラクタを使って `Uri.Builder` オブジェクトを生成する
2. `Builder#path` メソッドを使って接続先 URL を設定する
3. `Builder#appendQueryParameter` メソッドを使用し WebAPI パラメータを設定する

```
Builder builder = new Builder();           ... 1
builder.path(URL);                          ... 2
builder.appendQueryParameter("title","android"); ... 3
```

手順 2. HttpGet オブジェクトの作成

クエリパラメータから `HttpGet` オブジェクトを生成します。

1. `Builder#build` メソッドを使用し、パラメータに基づいた `Uri` オブジェクトを生成する
2. `Uri#decode` メソッドを使用して、パラメータ付きの接続先 URL 文字列を生成する
3. 生成した URL を引数に指定し、`HttpGet` オブジェクトを生成する

```
String uri = Uri.decode(builder.build().toString()); ... 1,2
HttpGet get = new HttpGet(uri);                     ... 3
```

手順 3. リクエストの送信

リクエストの送信は実習 2 までの手順と同じです。

```
HttpResponse res = client.execute(get);
```

リスト 8.4: 手順 1 から 3 をまとめたコード

```
1: Builder builder = new Builder();
2: builder.path(URL);
3: builder.appendQueryParameter("title", "android");
4:
5: String uri = Uri.decode(builder.build().toString());
6: HttpGet get = new HttpGet(uri);
```

アプリケーションの実行

アプリケーションを実行し、リクエストに対応するレスポンスデータが帰ってくることを確認します。今回の例では、国会図書館に対し、書籍タイトルに "android" が含まれているものの検索リクエストを実行しました。検索結果が xml 形式で返ってきます。

8.6 【実習】HTTP 通信 3



HTTP通信 3

- 実習3
 - 前回の実習で作成したプログラムに、WebAPIを使った機能を追加する。



値を入力後クリック





実習 3 WebAPI を使った通信

実習 2 で作成したプログラムに、国会図書館 API を使って書籍データの検索機能を追加しましょう。

国会図書館の WebAPI 概要

国会図書館の OpenSearch の WebAPI を使って書籍データを取得します。

- リクエスト URL
 - <http://iss.ndl.go.jp/api/opensearch>

手順

1. リソースファイルの修正
2. Activity クラスの修正

手順 1. リソースファイルの修正

activity_http_sample.xml を修正し、画面に WebAPI を入力できるようにします。

リスト 8.5: activity_http_sample.xml

```
1: <!-- TODO 【HTTP 通信 実習 3】 WebAPI パラメータレイアウトを追加する -->
2:
3: <LinearLayout
4:     android:layout_width="match_parent"
5:     android:layout_height="wrap_content" >
6:
7:     <TextView
8:         android:layout_width="50dp"
9:         android:layout_height="wrap_content"
10:        android:text="@string/key" />
11:
12:    <EditText
13:        android:id="@+id/editKey"
14:        android:layout_width="match_parent"
15:        android:layout_height="wrap_content" >
16:    </EditText>
17: </LinearLayout>
18:
19: <LinearLayout
20:     android:layout_width="match_parent"
21:     android:layout_height="wrap_content" >
22:
23:     <TextView
24:         android:layout_width="50dp"
25:         android:layout_height="wrap_content"
26:         android:text="@string/value" />
27:
28:     <EditText
29:         android:id="@+id/editValue"
30:         android:layout_width="match_parent"
31:         android:layout_height="wrap_content" >
32:     </EditText>
33: </LinearLayout>
```

手順 2. Activity クラスの修正

接続先 URL の変更し、WebAPI パラメータ対応の処理を追加します。取得結果の文字列は UTF で取得します。 実習 1,2 で作成した HttpGet オブジェクトの処理はコメントアウトします。

- 接続先
 - <http://iss.ndl.go.jp/api/opensearch>
- WebAPI パラメータ対応
 - URL の末尾にの ? 以降の値を付与してリクエストを投げる

● 接続先例

– <http://iss.ndl.go.jp/api/opensearch?title=android>

リスト 8.6: HttpSampleActivity

```
1: public class HttpSampleActivity extends Activity {
2:
3:     private static final String TAG = "HttpClientSample";
4:     // 接続先 URL ネットワーク接続可
5:     // private static final String URL = "http://www.oesf.jp/";
6:
7:     // 実習 3 ネットワーク接続可
8:     private static final String URL = "http://iss.ndl.go.jp/api/opensearch";
9:
10:    ...
11:
12:    public void onClickButton(View v) {
13:        // TODO 【HTTP 通信 実習 1】No.01 DefaultHttpClient オブジェクトの生成する
14:        DefaultHttpClient client = new DefaultHttpClient();
15:
16:        // TODO 【HTTP 通信 実習 3】No.01 WebAPI 用クエリパラメータの作成
17:        EditText editKey = (EditText) findViewById(R.id.editKey);
18:        EditText editValue = (EditText) findViewById(R.id.editValue);
19:        Builder builder = new Builder();
20:        builder.path(URL);
21:        builder.appendQueryParameter(editKey.getText().toString(), editValue
22:            .getText().toString());
23:
24:        // TODO 【HTTP 通信 実習 3】No.02 作成したクエリパラメータから HttpGet オブジェクト
25:        // を生成する
26:        String uri = Uri.decode(builder.build().toString());
27:        HttpGet get = new HttpGet(uri);
28:
29:        // TODO 【HTTP 通信 実習 1】No.02 GET メソッドで接続するリクエストオブジェクトを生成
30:        // する
31:        //HttpGet get = new HttpGet(URL);
32:
33:        try {
34:            ...
35:
36:            // TODO 【HTTP 通信 実習 2】 No.01 html 文の取得
37:            String content = EntityUtils.toString(res.getEntity());
38:            ...
39:        }
40:    }
```

確認

アプリケーションを実行し、取得したレスポンスデータが TextView に表示されることを確認しよう。

パラメータには表 8.3 のようなものが使用できます。

パラメータの詳細については "<http://iss.ndl.go.jp/information/api/>" を参照してください。

表 8.3 パラメータ設定情報

Key	Value
title	書籍のタイトルに含まれる文字列
creator	著者名



解答

解答ドキュメントを参照してください。

第9章

JSON 解析

この章の目的

- JSON の概要を理解する
- JSON の解析方法を理解する

9.1 JSON

JSON とは

JSON とは JavaScript Object Notation の略で、もともとは JavaScript でオブジェクトの記述方法です。データの表現が容易であることから、JavaScript 以外にもさまざまなところで使われています。JSON には次のような特徴があります。

JSON の特徴

- 軽量のデータ記述言語
- マークアップ言語ではないので、XML の解析より高速に処理することが可能
- キー・バリューペアの集まりでデータ構造を記述している
- 文字コードは基本的に UTF-8 のみ

JSON の構造

JSON 形式のデータは、名前と値のペアの集合になります。たとえば、次のようなデータがあったとします。

- 名前：山田太郎
- 年齢：35
- メールアドレス：taro@example.com

このデータは、次のように表現できます。

```
{ "name" : "山田太郎", "age" :35, "mail" : "taro@example.com" }
```

「{」から「}」までで1つのオブジェクトを表します。オブジェクト内には、キーとバリューのペアを記述します。

JSON の書式

JSON フォーマットは次のように記述します。

```
{ "key":value , "key":value, ... }
```

オブジェクト:

「{」から始まり、「}」で終わります。オブジェクト内には key と value のペアを記述します。それぞれのペアはカンマ(,)で区切ります。key は必ず二重引用符で囲みます。value には他のオブジェクト、文字列、数値、真偽値、配列を指定できます。また、null を指定することも可能です。

文字列:

「"name":"田中次郎"」のように、二重引用符で囲んで指定します。

数値:

「"age":35」のように、10進数の整数または浮動小数点数を指定します。

真偽値:

「"isStudent":true」のように、小文字で true または false と指定します。

配列:

「[要素, 要素, ...]」のように0個以上の要素をカンマ(,)で区切って指定します。配列はつぎのように記述します。

配列の例

```
["ABC", "DEF", "GHI"]
[10, 20, 30]
[
  { "name" : "山田太郎", "age" : 35 , "mail" : "taro@example.com" },
  { "name" : "鈴木花子", "age" : 28 , "mail" : "hanako@example.com" },
  { "name" : "田中次郎", "age" : 42 , "mail" : "jiro@example.com" }
]
```


JSON の解析

Android では `json.org` パッケージを利用して JSON 形式のデータを扱うことができます。次のような JSON を解析します。この JSON から `"value1"`, `"value2"` を取得します。

```
{
  "root": {
    "node": [
      "value1",
      "value2"
    ]
  }
}
```

手順

1. JSON 文字列の構造分析する
2. JSON 文字列から JSON のオブジェクトを生成する
3. `JSONObject` または `JSONArray` からデータを取得する

手順 1. JSON 文字列の構造分析する

JSON 文字列の構造を十分に把握し、ルートデータが `JSONObject` なのか、`JSONArray` なのかを確認します。

ルートの型:

`JSONObject` です。 `"root"` という key を持っていて value の型は `JSONObject` です。

root:

`JSONObject` です。 `"root"` という key を持っていて value の型は `JSONArray` です。

node:

`String` 配列です。 `"value1"`, `"value2"` を持っています。

手順 2. JSON 文字列から JSON のオブジェクトを生成する

`JSONObject` または `JSONArray` のコンストラクタに JSON 文字列を引数で指定してオブジェクトを生成する今回のケースではルートの型が `JSONObject` なので、`JSONObject` で初期化します。

```
JSONObject json = new JSONObject(strJson);
```

手順 3. JSONObject または JSONArray からデータを取得する

JSONObject と JSONArray は型に応じた getter メソッドが用意されています。キーまたは、要素番号を引数に指定してデータを取得します。

表 9.1 JSONObject と JSONArray の主な getter メソッド

型	メソッド
int	getInt
double	getDouble
String	getString
JSONArray	getJSONArray
JSONObject	getJSONObject

今回のケースでは次のような処理になります。

1. JSONObject から "root" をキーに指定し JSONObject を生成する
2. 生成した JSON を Object から "node" をキーに指定し、JSONArray を生成する
3. JSONArray を for で繰り返し、要素番号を指定して n 番目のデータを取得する

```
ArrayList<String> values = new ArrayList<String>();
try {
    JSONObject json = new JSONObject(strJson);
    JSONObject root = json.getJSONObject("root");
    JSONArray node = root.getJSONArray("node");

    for(int i = 0; i < node.length(); i++){
        values.add(node.getString(i));
    }
} catch (JSONException e) {
    Log.e("JsonSampleForDocActivity" , e.getMessage(), e);
}
```

9.2 【実習】JSON

**実習**

次のような JSON を解析するアプリケーションを作成しましょう。JSON が記述されたファイルは assets 以下にファイル名「sample.json」で保存されています。

```

{
  "feed": {
    "entries": [
      {
        "id": "hPzNl6NKAG0",
        "group": {
          "description": "全てはより良い滑り込みのために。",
          "thumbnail_url": "image/cat.jpg",
          "title": "特訓するねこ。"
        }
      },
      {
        "id": "xxxxxxxx",
        "group": {
          "description": "xxxxxxxx",
          "thumbnail_url": "xxxxxxxx",
          "title": "xxxxxxx"
        }
      }
    ]
  }
}

```

※解析対象

図: sample.json

プロジェクト概要

実習用スケルトンプロジェクトを Eclipse に取り込みます。

プロジェクト名 : 「JsonSample_skeleton」

表 9.2 プロジェクト概要

項目	設定値
Project Name	JSONSample
Build Target	4.4
Aplication name	JSONSample
Package	com.example.listsample
Create Activity	JSONSampleActivity
Layout File	activity_sample_json.xml

手順

未実装の処理を実装して、アプリケーションを完成させましょう。

1. リソースファイルの修正 <実装済>
2. データクラス Item の作成 <実装済>
3. JSON 解析クラス JSONHelper の作成 <未実装>
4. Activity に Button 押下時の処理を追加する <実装済>

手順 1. リソースファイルの修正 <実装済>

リソースファイルを修正し、画面に Button と TextView を配置します。

リスト 9.1: activity_json_sample.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <Button
6:         android:id="@+id/button_clear"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content"
9:         android:onClick="onClickClearButton"
10:        android:text="Clear" >
11:    </Button>
12:
13:    <Button
14:        android:id="@+id/button_json"
15:        android:layout_width="match_parent"
16:        android:layout_height="wrap_content"
17:        android:onClick="onClickJsonButton"
18:        android:text="JSON" >
19:    </Button>
20:
21:    <TextView
22:        android:id="@+id/text_result"
23:        android:layout_width="wrap_content"
24:        android:layout_height="match_parent"
25:        android:inputType="textMultiLine" >
26:    </TextView>
27:
28: </LinearLayout>
```

手順 2. データクラス Item の作成 <実装済>

JSON データから次の要素を取り出しデータクラス化した Item クラスを作成します。

要素に対応するメンバ変数

id:

String id

title:

String title

thumbnail_url:

String thumbnail_url

description:

String description

リスト 9.2: Item.java

```
1: package com.example.jsonsample;
2:
3: public class Item {
4:     public String id;
5:     public String title;
6:     public String description;
7:     public String thumbnail_url;
8:
9:     @Override
10:    public String toString() {
11:        return "Item [id=" + id + ", title=" + title + ", description=" + description
12:            + ", thumbnail_url=" + thumbnail_url + "];"
13:    }
14:
15: }
```

手順 3. JSON 解析クラス JSONHelper の作成 <未実装>

JSON データを解析するためのロジッククラス JSONHelper クラスが作成されています。このクラスは次のメソッドが定義されています。

```
public static ArrayList<Item> parseJson(String strJson)
public static Item parseToItem(JSONObject json)
```

この2つのメソッドは定義のみされており、処理は実装されていません。未実装の処理を実装させます。

- parseJson
 - 引数で受け取った JSON 文字列を解析する
 - "entries" キーから JSONArray を取得し、要素ごとに繰り返し、parseToItem を呼び出す
- parseToItem
 - 引数で受け取った JSONObject からデータを取り出し、値を Item にセットする

リスト 9.3: JSONHelper.java

```
1: package com.example.jsonsample;
2:
3: import java.util.ArrayList;
4:
5: import org.json.JSONArray;
6: import org.json.JSONException;
```

```
7: import org.json.JSONObject;
8:
9: import android.util.Log;
10:
11: public class JsonHelper {
12:
13:     public static ArrayList<Item> parseJson(String strJson) {
14:         ArrayList<Item> list = new ArrayList<Item>();
15:         try {
16:             // TODO JSON 解析
17:             JSONObject json = new JSONObject(strJson);
18:             JSONObject feed = json.getJSONObject("feed");
19:             JSONArray entries = feed.getJSONArray("entries");
20:             for (int i = 0; i < entries.length(); i++) {
21:                 JSONObject entry = entries.getJSONObject(i);
22:                 list.add(parseToItem(entry));
23:             }
24:         } catch (Exception e) {
25:             Log.e("JsonHelper", e.getMessage(), e);
26:         }
27:
28:         return list;
29:     }
30:
31:     public static Item parseToItem(JSONObject json) throws JSONException {
32:         Item item = new Item();
33:         // TODO JSON 解析 entries キーから取り出した JSONObject を RowModel にパースする
34:
35:         item.id = json.getString("id");
36:         JSONObject group = json.getJSONObject("group");
37:         item.title = group.getString("title");
38:         item.description = group.getString("description");
39:         item.thumbnail_url = group.getString("thumbnail_url");
40:
41:         Log.v("JsonHelper", item.toString());
42:         return item;
43:     }
44: }
```

手順 4. Activity に Button 押下時の処理を追加する <実装済>

Json ボタン押下時に `JsonHelper#parseJson` メソッドを呼び出し、解析後のデータリストを取得します。TextView に解析結果を表示します。

リスト 9.4: `JsonSampleActivity.java`

```
1: public void onClickJsonButton(View v) {
2:     ArrayList<Item> list = JsonHelper.parseJson(json);
3:     StringBuilder sb = new StringBuilder();
4:     for (Item item : list) {
5:         sb.append(item.toString());
6:     }
```

```
7:     textResult.setText(sb.toString());
8: }
```

確認

アプリケーションを実行し、図 9. のように表示されることを確認します。



解答

解答ドキュメントを参照してください。

【補足実習】

asset/sample.json を編集して entries 内にデータを追加し、2 つ以上のデータクラスを取得できることを確認します。


```
{
  "feed": {
    "entries": [
      {
        "id": "hPzNl6NKAG0",
        "group": {
          "description": "全てはより良い滑り込みのために。",
          "thumbnail_url": "image/cat.jpg",
          "title": "特訓するねこ。"
        }
      },
      {
        "id": "xxxxxxxx",
        "group": {
          "description": "xxxxxxxx",
          "thumbnail_url": "xxxxxxxx",
          "title": "xxxxxx"
        }
      }
    ]
  }
}
```

※データを追加する

第 10 章

データベース

この章の目的

- SQLite の概要について理解する
- adb コマンドを理解する
- Android からデータベースを操作する方法を習得する

10.1 SQLite

SQLite とは

軽量データベースです。データ保存に単一ファイルを利用します。データベースの操作に SQL を使用でき、トランザクションを管理できます。仕様、ソースコードが公開されていて、多くの言語でドライバが開発されています。

SQLite のデータ型

SQLite は次の 5 種類のデータ型に対応されています。これは SQLite が実際に扱うデータ型であって、カラムに指定するデータ型ではありません。

表 10.1 SQLite のデータ型

型	設定値
TEXT	テキスト。文字列
INTEGER	整数値
REAL	浮動小数点
BLOB	バイナリデータ
NULL	NULL

カラムに指定するデータ型

SQLite はカラムにデータ型を定義する必要はありません。

```
CREATE TABLE sample_table(_id,name,value);
```

カラムに型を指定する場合は、次のようなデータ型を指定することができます。

- TEXT
- NUMERIC
- INTEGER
- REAL
- NONE

カラムのデータ型と値のデータ型は次のような組み合わせになります。

表 10.2 カラムに指定するデータ型

型	設定値
TEXT	全てのデータ型を TEXT として扱う。
NUMERIC	TEXT と同じだが、数値は INTEGER または REAL に振り分けられる
INTEGER	NUMERIC と同じだが、小数点以下が 0 の場合は整数値として表示
REAL	NUMERIC と同じだが、小数点以降も表示する
NONE	データの型変換を行わない

```
CREATE TABLE sample_table(_id INTEGER, name TEXT, value INTEGER);
```

SQLite では、データ追加時にカラムのデータ型と値のデータ型に合わせた自動的な型変換が行われるため、どのような型でもデータを追加することができます。そのため、カラムに対し別の型を指定しても仕様上エラーにはなりません。そのため、プログラム側でデータ型を意識する必要があります。

Android で SQLite を操作する

Android は標準で SQLite がサポートされています。Android から SQLite を操作するには次のクラスを使用します。

SQLiteOpenHelper クラス:

データベースを作成、データベースとの接続（オープン）、切断（クローズ）を行う

SQLiteDatabase クラス:

テーブルの検索、追加、更新、削除を行う。指定した SQL クエリを実行する

データベースを作成する

データベースを作成するには、SQLiteOpenHelper クラスを継承するサブクラスを作成します。作成したサブクラスのコンストラクタで SQLiteOpenHelper クラスのコンストラクタを実行すると、SQLite のデータベースファイルが作成されます。データベースファイルが既に存在する場合、2 回目以降のコンストラクタが実行されてもファイルが作成されません。ファイルが存在しない場合のみデータベースファイルが作成されます。

テーブルを作成する

SQLiteOpenHelper を使用してデータベースが作成されると、onCreate メソッドが実行されます。onCreate メソッド内でテーブルを作成する SQL を実行し、テーブルを作成します。SQL を実行するためには、SQLiteDatabase オブジェクトが必要ですが、onCreate メソッドの引数で与えられます。与えられた SQLiteDatabase オブジェクトを使用して SQL を実行し、テーブルを作成します。

データベースとテーブルを作成する

SQLiteOpenHelper と SQLiteDatabase クラスを使って、データベースとテーブルを作成してみましょう。

手順

1. SQLiteOpenHelper クラスのサブクラスを作成する
2. コンストラクタで、データベースを作成する
3. SQLiteOpenHelper の抽象メソッドをオーバーライドする
4. SQLiteOpenHelper#onCreate メソッドで、テーブルを生成する SQL 文(CREATE TABLE 文)を実行する
5. Activity に データベース接続する処理を追加する
6. データベースを閉じる

手順 1. SQLiteOpenHelper クラスのサブクラスを作成する

SQLiteOpenHelper クラスを継承した独自の SQLiteOpenHelper クラスを作成します。

```
public class SampleSQLiteOpenHelper extends SQLiteOpenHelper {
```

手順 2. コンストラクタで、データベースを作成する

コンストラクタで SQLiteOpenHelper クラスのコンストラクタを実行し、データベースを作成します。第 1 引数に Context、第 2 引数にデータベース名、第 3 引数に CursorFactory、第 4 引数にデータベースのバージョンを指定します。

```
public SampleSQLiteOpenHelper(Context context) {
    // データベースを作成する (データベース名は「 SAMPLE_DATABASE 」)
    super(context, "SAMPLE_DATABASE", null, 1);
}
```

手順 3. SQLiteOpenHelper の抽象メソッドをオーバーライドする

onCreate メソッドと onUpgrade メソッドをオーバーライドします。

onCreate:

テーブルを作成する処理を実装するメソッド

onUpgrade:

テーブル定義を更新する処理を実装するメソッド (コンストラクタの第 4 引数の値が変更されると、このメソッドが呼び出されます。)

手順 4. SQLiteOpenHelper#onCreate メソッドで、テーブルを生成する SQL 文 (CREATE TABLE 文) を実行する

テーブルを生成する SQL 文の文字列を作成します。onCreate メソッド第一引数 SQLiteDatabase database の execSQL メソッドに SQL 文の文字列を与え、実行します。

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE SAMPLE_TABLE(_id INTEGER,name TEXT);");
}
```

Primary Key は "_id" という名称で定義する

Android では、次の理由により、テーブルの Primary Key となるカラムは "_id" という名称で定義する必要があります。

理由

- Content Provider 経由でデータベースを利用する際に "_id" という列の値をユニーク ID とする仕様のため
- CursorAdapter、ArrayListCursor 等のクラスで、"_id" という列から情報を取得する実装が存在するため

手順 1 から 4 をまとめたコード

```
public class SampleSQLiteOpenHelper extends SQLiteOpenHelper {

    public SampleSQLiteOpenHelper(Context context) {
        super(context, "SAMPLE_DATABASE", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE SAMPLE_TABLE(_id INTEGER,name TEXT);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }

}
```

5. Activity に データベース接続する処理を追加する

作成した SQLiteOpenHelper を使って、データベースの接続を行います。Activity で SQLiteOpenHelper のインスタンスを生成します。初めて SQLiteOpenHelper のコンストラクタが実行される時はデータベースファイルが作成され、テーブルが作成されます。2 回目以降はインスタンスの生成のみ行います。

```
SampleSQLiteOpenHelper helper = new SampleSQLiteOpenHelper(this);
```

生成されたインスタンスを使ってデータベースに接続するためのメソッドを実行します。接続用のメソッドには次のようなメソッドがあります。

getReadableDatabase:

読み込み専用で SQLiteDatabase オブジェクトを取得する

getWritableDatabase:

書き込み可能な SQLiteDatabase オブジェクトを取得する

ここでは onCreate メソッドで、読み込み専用で接続します。

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        SampleSQLiteOpenHelper helper = new SampleSQLiteOpenHelper(this);
        SQLiteDatabase database = helper.getReadableDatabase();
        if(database != null){
            Log.v("DatabaseSample", "Succeeded in open the database.");
        }
    }

    ...
}
```

6. 不要になったタイミングでデータベースを閉じる

不要になったタイミングでデータベースを閉じます。データベースを閉じるには、SQLiteOpenHelper クラスの close メソッドを実行します。

```
helper.close();
```

SQLiteDatabase クラスの isOpen メソッドを使うと、データベースが閉じているか確認できます。

```
if(!database.isOpen()){
    Log.v("DatabaseSample", "Succeeded in close the database.");
}
```

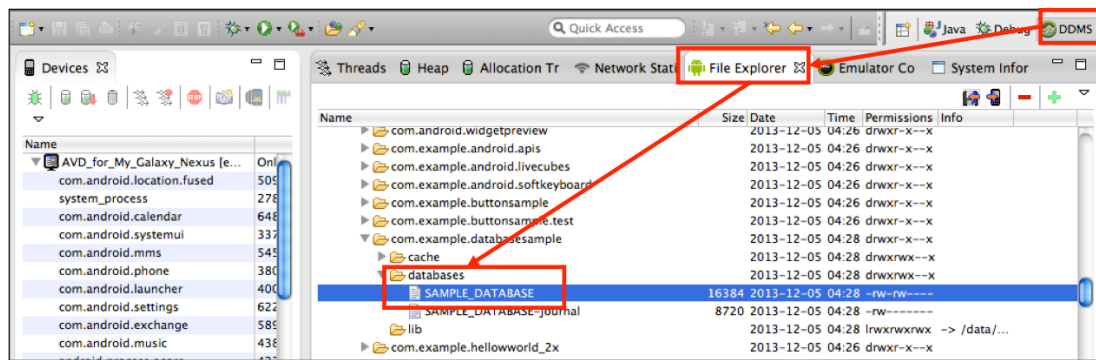
アプリケーションを実行する

アプリケーションを実行し、次のことを確認します。

データベースが作成されていることを確認する

DDMS を起動しファイルエクスプローラー「com.example.databasesample」の「database」内に「SAMPLE_DATABASE」が作成されていることを確認します。

データベースファイルはデフォルトでは /data/data/アプリケーションのパッケージ名/databases 以下に保存されます。



ログが出力されていることを確認する

データベースの作成とクローズログが出力されていることを確認します。

V	12-05	04:28:50.611	775	775	com.example.databasesample DatabaseSample	Succeeded in open the database.
V	12-05	04:28:50.611	775	775	com.example.databasesample DatabaseSample	Succeeded in close the database.

10.2 【実習】データベース 1



実習 1 データベースの作成

データベースの作成と接続をするアプリケーションを作りましょう。

プロジェクト概要

表 10.3 プロジェクト概要

項目	設定値
Project Name	DatabaseSample
Build Target	4.4
Aplication name	DatabaseSample
Package	com.example.databasesample
Create Activity	DatabaseSampleActivity
Layout File	activity_database_sample

手順

1. SQLiteOpenHelper のサブクラスを作成する
2. Activity にデータベースの接続処理を追加する

手順 1. SQLiteOpenHelper のサブクラスを作成する

SQLiteOpenHelper を継承した SampleSQLiteOpenHelper を作成します。次のような設定で、データベースとテーブルを作成します。

<データベース概要>**データベース名:**

SAMPLE_DATABASE

テーブル名:

SAMPLE_TABLE

<テーブル構造>**_id:**

INTEGER 型、主キー、auto increment

name:

TEXT 型、Not Null

value:

INTEGER 型、Not Null

1. 定数の作成

データベース名、テーブル名、テーブル作成クエリを定数で用意します。

```
public static final String SAMPLE_DATABASE = "SAMPLE_DATABASE";
public static final String SAMPLE_TABLE = "SAMPLE_TABLE";
public static final String CREATE_TABLE = "CREATE TABLE " + SAMPLE_TABLE
    + "(_id INTEGER PRIMARY KEY AUTOINCREMENT" + ",name TEXT not null"
    + ",value INTEGER not null" + ");";
```

2. コンストラクスの作成

コンストラクタで親クラスのコンストラクタを呼び出し、データベースを作成します。

```
public SampleSQLiteOpenHelper(Context context) {
```

```
    super(context, SAMPLE_DATABASE, null, 1);
}
```

3. onCreate、onUpgrade のオーバーライド

onCreate メソッドでテーブルを作成するクエリを実行し、onUpgrade メソッドはから実装します。

```
@Override
public void onCreate(SQLiteDatabase database) {
    database.execSQL(CREATE_TABLE);
}

@Override
public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
}
```

手順 1 をまとめたコード

リスト 10.1: SampleSQLiteOpenHelper.java

```
1: package com.example.databasesamle;
2:
3: import android.content.Context;
4: import android.database.sqlite.SQLiteDatabase;
5: import android.database.sqlite.SQLiteOpenHelper;
6:
7: public class SampleSQLiteOpenHelper extends SQLiteOpenHelper {
8:
9:     public static final String SAMPLE_DATABASE = "SAMPLE_DATABASE";
10:    public static final String SAMPLE_TABLE = "SAMPLE_TABLE";
11:    public static final String CREATE_TABLE = "CREATE TABLE " + SAMPLE_TABLE
12:        + "(_id INTEGER PRIMARY KEY AUTOINCREMENT" + ",name TEXT not null"
13:        + ",value INTEGER not null" + ");";
14:
15:    public SampleSQLiteOpenHelper(Context context) {
16:        super(context, SAMPLE_DATABASE, null, 1);
17:    }
18:
19:    @Override
20:    public void onCreate(SQLiteDatabase database) {
21:        database.execSQL(CREATE_TABLE);
22:    }
23:
24:    @Override
25:    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
26:    }
27:
```

```
28: }
```

手順 2. Activity にデータベースの接続処理を追加する

onCreate メソッドでデータベースの接続処理を追加します。SampleSQLiteOpenHelper を使って、SQLiteDatabase オブジェクトを読み込み専用で取得し、接続されていることを確認しクローズします。接続とクローズのタイミングでログを出力します。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_database_sample);
    SampleSQLiteOpenHelper helper = new SampleSQLiteOpenHelper(this);
    SQLiteDatabase database = helper.getReadableDatabase();
    if(database != null && database.isOpen()){
        Log.v("DatabaseSample", "Succeeded in open the database.");
        helper.close();
        Log.v("DatabaseSample", "Succeeded in close the database.");
    }
}
```

確認

アプリケーションを実行し、次のことを確認します。

- データベースが作成されている
- 接続とクローズログが出力されている



V	12-05 04:28:50.611	775	775	com.example.databasesample DatabaseSample	Succeeded in open the database.
V	12-05 04:28:50.611	775	775	com.example.databasesample DatabaseSample	Succeeded in close the database.

解答

解答ドキュメントを参照してください。

10.3 sqlite3

sqlite3 とは

sqlite3 を使うことで、コマンドラインからデータベースを操作することができます。sqlite3 はシェルを起動して利用します。

シェルの起動する

adb shell コマンドは、端末・エミュレータに接続し、シェルを起動します。

```
> adb shell
```

データベースに接続する

シェル画面から sqlite3 <データベースのパス> でデータベースに接続することができます。

```
# sqlite3 /data/data/com.example.databasesample/databases/SAMPLE_DATABASE
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

sqlite3 コマンド

sqlite3 では次のようなコマンドを実行することができます。

.help:

ヘルプを表示する

.dump:

ダンプファイルを表示する

.schema:

CREATE 文を表示する

.tables:

テーブル一覧を表示する

SQL 文:

SQL 文を発行する。

例) SELECT * FROM SAMPLE_TABLE;

.header on | off:

カラム名の表示を ON/OFF にする

.exit:

sqlite3 を終了する

10.4 【実習】データベース 2



データベース 2

- 実習 2
 - sqlite3 コマンドを使って、実習 1 で作成したデータベースを確認する

出力例)

```
$ adb shell
# cd /data/data/com.example.databasesample/databases
# sqlite3 SAMPLE_DATABASE
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
SAMPLE_TABLE  android_metadata
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE android_metadata (locale TEXT);
INSERT INTO "android_metadata" VALUES('en_US');
CREATE TABLE SAMPLE_TABLE(_id INTEGER,name TEXT);
COMMIT;
sqlite>
```

実習 2 SQLite

sqlite3 を使ってデータベースに接続しましょう。

手順

1. シェルを起動する
2. データベースファイルを確認する
3. sqlite3 コマンドでデータベースに接続する
4. テーブルの CREATE 文を表示する

手順 1. シェルを起動する

adb コマンドを使ってシェルを起動します

```
> adb shell
```


手順 2. データベースファイルを確認する

データベースファイルのディレクトリまで移動し、データベースファイルが存在することを確認します。

```
# cd /data/data/com.example.databasesample/databases/  
# ls  
SAMPLE_DATABASE
```

手順 3. sqlite3 コマンドでデータベースに接続する

データベースに接続します。

```
# sqlite3 SAMPLE_DATABASE  
SQLite version 3.7.11 2012-03-20 11:35:50  
Enter ".help" for instructions  
Enter SQL statements terminated with a ";"  
sqlite>
```

手順 4. テーブルの CREATE 文を表示する

テーブルが存在することを確認し、CREATE 文を表示します。

```
sqlite> .tables  
SAMPLE_TABLE      android_metadata  
  
sqlite> .dump  
PRAGMA foreign_keys=OFF;  
BEGIN TRANSACTION;  
CREATE TABLE android_metadata (locale TEXT);  
INSERT INTO "android_metadata" VALUES('en_US');  
CREATE TABLE SAMPLE_TABLE(_id INTEGER,name TEXT);  
COMMIT;
```

確認

CREATE 文が正しく表示されていることを確認します。

CREATE 文が間違っていた場合

テーブル作成が間違っていた場合は次の手順で修正します。

1. DDMS からデータベースファイルを削除する
2. テーブル作成部分のソースを修正し、アプリケーションを実行する
3. データベースの確認をする

10.5 データの検索

データを検索する方法

データの検索は `SQLiteDatabase#query` メソッドで行います。query メソッドは引数に `SELECT` 文で使う要素を指定します。戻り値の `Cursor` は結果セットをループしてデータを取得したり、Index 指定で取得することができます。

`Cursor` の使用方法については後述

```
public Cursor query(String table,
                   String[] columns,
                   String selection,
                   String[] selectionArgs,
                   String groupBy,
                   String having,
                   String orderBy,
                   String limit )
```

表 10.4 query メソッドの引数

引数	説明
String table	テーブル名
String[] columns	取得する列名を配列で指定する
String selection	検索条件
String[] selectionArgs	selection に ? が含まれていた場合、置換する値を配列で指定する
String groupBy	GROUP BY 句相当の文字列
String having	HAVING 句相当の文字列
String orderBy	ORDER BY 句相当の文字列
String limit	LIMIT 句相当の文字列

全件検索する

検索条件や取得配列を指定しない場合は、query メソッドの第 2 引数以降を `null` で指定します。戻り値の `Cursor` オブジェクトから `getCount` メソッドを使用することで件数を取得することができます。`Cursor` は、不要になったタイミングで閉じる必要があります。`Cursor#close` メソッドを使って `Cursor` を閉じます。

リスト 10.6: 全件検索の例

```
1: SampleSQLiteOpenHelper helper = new SampleSQLiteOpenHelper(this);
2: SQLiteDatabase database = helper.getReadableDatabase();
3:
```

```
4: // 全件検索する
5: Cursor cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, null,
6:     null, null, null, null);
7: if (cursor != null) {
8:     Log.v("DatabaseSample", "[データ件数:" + cursor.getCount() + "件]");
9:     cursor.close();
10: }
11: helper.close();
```

10.6 【実習】データベース3



実習3 データの全件検索

前回の実習で作成したプログラムに、全件検索機能を追加します。

手順

1. リソースファイルの修正
2. 検索結果画面の作成
3. 検索ボタン押下時の処理追加
4. 検索結果画面に検索処理の追加

手順1. リソースファイルの修正

リソースファイルを次のように修正します。

strings.xml

文字列リソースを追加します

リスト 10.3: strings.xml

```
1: <string name="search">検索</string>
```

メイン画面のレイアウト

メイン画面のレイアウトに Button を追加します。

リスト 10.4: activity_database_sample.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <Button
6:         android:id="@+id/button1"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content"
9:         android:text="@string/search"
10:        android:onClick="onClickSearchButton" />
11:
12: </LinearLayout>
```

検索結果画面のレイアウト

検索結果画面のレイアウトを次のように設定します。

ファイル名: activity_result.xml

リスト 10.5: activity_result.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView
6:         android:id="@+id/text_count"
7:         android:layout_width="wrap_content"
8:         android:layout_height="wrap_content" />
9:
10: </LinearLayout>
```

手順 2. 検索結果画面の作成

検索結果画面の Activity を作成し、AndroidManifest.xml ファイルに登録します。

ファイル名: ResultActivity.java

手順 3. 検索ボタン押下時の処理追加

DatabaseSampleActivity で検索ボタンが押されたときに検索結果画面へ遷移する処理を追加する。

```
public void onClickSearchButton(View v) {
    Intent intent = new Intent(this, ResultActivity.class);
    startActivity(intent);
}
```

手順 4. 検索結果画面に検索処理の追加

ResultActivity クラスに検索処理と、Cursor のクローズ処理を追加します。

1. メンバ変数の定義

Cursor 型のメンバ変数 cursor を定義します。

```
Cursor cursor;
```

2. 検索処理を追加する

onCreate メソッドに検索処理を追加します。SQLiteDatabase#query を実行し Cursor を取得します。Cursor から結果件数を取得し、text_search に次のメッセージを表示させます。

```
[データ件数 : XX 件]
```

DatabaseSampleActivity の onCreate メソッドに記述した DB 接続処理のコードは削除します。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_result);
    TextView textCount = (TextView) findViewById(R.id.text_count);

    SampleSQLiteOpenHelper helper = new SampleSQLiteOpenHelper(this);
    SQLiteDatabase database = helper.getReadableDatabase();

    // 全件検索する
```

```
cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, null, null, null,
    null, null);
if (cursor != null) {
    textCount.setText("[データ件数:" + cursor.getCount() + "件]");
}
helper.close();
}
```

3. Cursor をクローズする

onDestroy メソッドで Cursor をクローズします。(ここでは Acitivity が破棄されるタイミングでクローズしている)

```
protected void onDestroy() {
    super.onDestroy();
    cursor.close();
}
```

確認

アプリケーションを実行し、検索件数が0件であることを確認します。



解答

解答ドキュメントを参照してください。

10.7 データの追加

データを追加する方法

データの追加は `SQLiteDatabase#insert` メソッドで行います。

```
public long insert (String table, String nullColumnHack, ContentValues values)
```

引数には追加したいレコード情報を指定します。戻り値は `_id` です。データの追加に失敗した場合は、`-1` が戻り値として返却されます。

`insert` メソッドのパラメータは次のようになります。

String table:

テーブル名

String nullColumnHack:

`null` が許容されていないカラムのデフォルト値

ContentValues values:

追加データのカラム名と値をもった `ContentValues` オブジェクト

リスト 10.6: `insert` の例

```
1: SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(this);
2: // 書込可能な SQLiteDatabase オブジェクトを取得する
3: SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();
4:
5: // insert データの設定
6: ContentValues values = new ContentValues();
7: values.put("name", "name1");
8: values.put("value", 100);
9:
10: // データを追加する
11: long result = database.insert(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, values);
12: Log.v("AddActivity", "Result:" + result);
13: // データベースから切断する
14: databaseOpenHelper.close();
15: if (result != -1) {
16:     Toast.makeText(this, "登録完了", Toast.LENGTH_LONG).show();
17: }
```

10.8 【実習】データベース4



実習4 データの追加

前回の実習で作成したプログラムを修正し、データの登録機能を追加しましょう。

手順

1. リソースファイルの修正
2. 登録画面を追加する
3. 登録メニュー押下時の処理を追加する
4. 登録処理を追加する

手順1. リソースファイルの修正

リソースファイルを次のように修正します。

strings.xml

文字列リソースを追加します

リスト 10.7: strings.xml

```
1: <string name="add">登録</string>
2: <string name="add_complete">登録完了</string>
3: <string name="name">Name</string>
4: <string name="value">Value</string>
```

追加画面のレイアウト

追加画面のレイアウトを次のようにします

リスト 10.8: activity_add.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView
6:         android:layout_width="wrap_content"
7:         android:layout_height="wrap_content"
8:         android:text="@string/name" />
9:
10:    <EditText
11:        android:id="@+id/edit_name"
12:        android:layout_width="match_parent"
13:        android:layout_height="wrap_content"
14:        android:ems="10" >
15:
16:    </EditText>
17:    <TextView
18:        android:layout_width="wrap_content"
19:        android:layout_height="wrap_content"
20:        android:text="@string/value" />
21:
22:    <EditText
23:        android:id="@+id/edit_value"
24:        android:layout_width="match_parent"
25:        android:layout_height="wrap_content"
26:        android:ems="10" >
27:
28:    </EditText>
29:
30:    <Button
31:        android:layout_width="match_parent"
32:        android:layout_height="wrap_content"
33:        android:text="@string/add"
34:        android:onClick="onClickAddButton" />
35:
36: </LinearLayout>
```

res/menu/datavase_sample.xml

メイン画面のメニューリソースを次のようにします。

リスト 10.9: datavase_sample.xml

```
1: <menu xmlns:android="http://schemas.android.com/apk/res/android" >
2:
3:     <item
4:         android:id="@+id/menu_add"
5:         android:title="@string/add">
6:     </item>
7:
8: </menu>
```

手順 2. 登録画面を追加する

登録画面の Activity を作成し、AndroidManifest.xml ファイルに登録します。
表示レイアウトに activity_add.xml を指定します。
ファイル名: AddActivity.java

手順 3. 登録メニュー押下時の処理を追加する

メイン画面に登録メニュー選択後に AddActivity に遷移する処理を追加する。

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Intent intent = new Intent(this, AddActivity.class);
    startActivity(intent);
    return false;
}
```

手順 4. 登録処理を追加する

AddActivity にデータの登録処理を追加します。

1. 登録ボタンが押されたときのメソッドを追加する

登録ボタンが押されると onClickAddButton メソッドが呼び出されます。メソッドの定義を追加します。

```
public void onClickAddButton(View v){
```

```
}
```

2. EditText の値を取得する

onClickAddButton メソッドに EditText の入力値を取得する処理を追加します。TextUtils クラスの isEmpty メソッドを使うと入力値のチェックができます。

```
// Name の取得
EditText editName = (EditText) findViewById(R.id.edit_name);
String name = editName.getText().toString();

// Value の取得
EditText editValue = (EditText) findViewById(R.id.edit_value);
String value = editValue.getText().toString();

if (!TextUtils.isEmpty(name) && !TextUtils.isEmpty(value)) {
}
}
```

3. データを登録する

EditText に入力された値を使ってデータの登録処理を追加します。登録結果の値を出力します。

```
SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(this);
// 書込可能な SQLiteDatabase オブジェクトを取得する
SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();

// insert データの設定
ContentValues values = new ContentValues();
values.put("name", name);
values.put("value", value);

// データを追加する
long result = database.insert(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, values);
Log.v("AddActivity", "Result:" + result);
// データベースから切断する
databaseOpenHelper.close();
```

4. 登録完了後の処理を追加する

登録完了後に Toast を表示し、Activity を終了します。

```
if (result != -1) {
    Toast.makeText(this, R.string.add_complete, Toast.LENGTH_LONG).show();
    finish();
}
```

手順 2 をまとめたコード

以上の処理を追加すると、onClickAddButton のコードは次のようになります。

```
public void onClickAddButton(View v) {

    // Name の取得
    EditText editName = (EditText) findViewById(R.id.edit_name);
    String name = editName.getText().toString();

    // Value の取得
    EditText editValue = (EditText) findViewById(R.id.edit_value);
    String value = editValue.getText().toString();

    if (!TextUtils.isEmpty(name) && !TextUtils.isEmpty(value)) {

        SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(this);
        // 書込可能な SQLiteDatabase オブジェクトを取得する
        SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();

        // insert データの設定
        ContentValues values = new ContentValues();
        values.put("name", name);
        values.put("value", value);

        // データを追加する
        long result = database.insert(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, values);
        Log.v("AddActivity", "Result:" + result);
        // データベースから切断する
        databaseOpenHelper.close();
        if (result != -1) {
            Toast.makeText(this, R.string.add_complete, Toast.LENGTH_LONG).show();
            finish();
        }
    }
}
```

確認

アプリケーションを実行し、次のことを確認します。

- insert メソッドの戻り値が -1 でないこと
- 登録完了メッセージの Toast が表示されていること

- 登録後に検索を行い、データ件数が1件になっていること
- Activity が終了し、検索画面が表示されていること



解答

解答ドキュメントを参照してください。

10.9 レコードの内容を取得する

データベースから取得したデータを取り出す

データベースから取得したデータを取り出すには `Cursor` インタフェースを使います。`Cursor` インタフェースは `SQLiteDatabase#query` メソッドの戻り値です。`Cursor` は検索結果の 1 レコードに対応しています。

Cursor インタフェースの使い方

`Cursor` インタフェースが提供しているメソッドを使うことにより、検索結果のカーソルを最初に移動したり、カーソルが指しているレコードから指定したデータを取り出すことができます。

Cursor インタフェースに定義されている主なメソッド

`int getCount():`

レコード件数を取得する

`int getInt(int columnIndex):`

`index` 指定したカラムの値を数値で取得する

`String getString(int columnIndex):`

`index` 指定したカラムの値を文字列で取得する

`int getColumnIndex(String columnName):`

指定したカラムの `index` 値を取得する

`boolean moveToFirst():`

先頭レコードにカーソルを移動する

`boolean moveToNext():`

次のレコードにカーソルを移動する

データを取り出す方法

手順


1. `SQLiteDatabase#query` で `Cursor` を取得する
2. 先頭レコードにカーソルを合わせる
3. カラムの `index` を取得する
4. カラム `index` を指定してデータを `Cursor` から取り出す

```
// 手順 1. SQLiteDatabase#query で Cursor を取得する
cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, null, null,
    null, null, null);
if (cursor != null) {
    textCount.setText("[データ件数: " + cursor.getCount() + "件");
}
```

```
// 手順 2. 先頭レコードにカーソルを合わせる
while (cursor.moveToNext()) {

    // 手順 3. カラムの index を取得する
    // 手順 4. カラム index を指定してデータを Cursor から取り出す
    String name = cursor.getString(cursor.getColumnIndex("name"));
    int value = cursor.getInt(cursor.getColumnIndex("value"));
    Log.v("ResultActivity", "name:" + name + " value:" + value);
}
}
```

10.10 【実習】データベース 5




データベース 5

- 実習 5
 - 取得データの表示
 - 前回の実習で作成したプログラムを修正し、検索結果をログに出力する

DatabaseSample

検索

データが1件以上
ある状態で検索実行



ResultActivity

[データ件数: 1件]

ログを確認する

```

D 12-05 05:37:35.790 8597 8597 com.example.databasesample grolloc_goldfish Emulator without GPU emulation detected.
D 12-05 05:37:40.571 8597 8593 com.example.databasesample deluikum CC_CONCURRENT Freed 78K, 7% free 2715K/29
V 12-05 05:37:49.591 8597 8597 com.example.databasesample ResultActivity name:name1 value:10
          
```

実習 5 取得データを表示する

前回の実習で作成したプログラムを修正し、検索結果をログに出力します。

手順

1. ResultActivity を修正する

手順 1. ResultActivity を修正する

取得したデータの内容をログに出力する処理を追加します。

```

protected void onCreate(Bundle savedInstanceState) {
    ...
    if (cursor != null) {
        textCount.setText("[データ件数: " + cursor.getCount() + "件");
        // データの数だけループする
        while (cursor.moveToNext()) {
          
```

```
// name を取得
String name = cursor.getString(cursor.getColumnIndex("name"));
// value を取得
int value = cursor.getInt(cursor.getColumnIndex("value"));
Log.v("ResultActivity", "name:" + name + " value:" + value);
}
}
helper.close();
}
```

確認

アプリケーションを実行し、取得データがログに出力されていることを確認します。



```
D 12-05 05:37:35.790 8597 8597 com.example.databasesample gralloc_goldfish Emulator without GPU emulation detected.
D 12-05 05:37:40.571 8597 8603 com.example.databasesample dalvikvm GC_CONCURRENT freed 78K, 7% free 2715K/29
V 12-05 05:37:49.591 8597 8597 com.example.databasesample ResultActivity name:name1 value:10
```

解答

解答ドキュメントを参照してください。

10.11 データを一覧表示する

一覧画面に表示する方法

Cursor から取得するデータベースのデータを画面に一覧表示するには、ListView と ListActivity クラスを使用します。Cursor の情報を一覧表示するために Adapter という仕組みを Android は提供しています。Cursor オブジェクトと ListActivity を Adapter を使用して関連付けを行うことにより、Cursor の情報を画面の ListView に表示します。

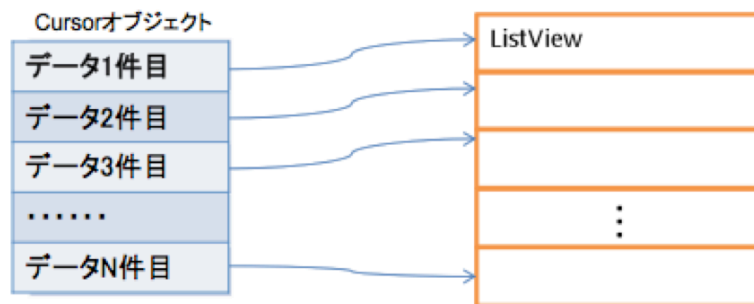


図: Cursor と ListView の関係

実装方法

Cursor オブジェクトの内容を一覧画面に表示するには、SimpleCursorAdapter オブジェクトを使います。SimpleCursorAdapter オブジェクトはコンストラクタを使って生成します。

```
public SimpleCursorAdapter (Context context,
                           int layout,
                           Cursor c,
                           String[] from,
                           int[] to,
                           int flags)
```

引数には次の値を指定します。

- 第 1 引数:Context
- 第 2 引数:行のレイアウト id
- 第 3 引数:カーソルオブジェクト
- 第 4 引数:カーソルから取得するデータのキー
- 第 5 引数:第 4 引数で指定したデータを表示させるビューの id
- 第 6 引数:アダプターの動作を指定するためのフラグ (CursorAdapter クラスに定数が提

供されています。)

ListActivity#setListAdapter メソッドを使用し、 SimpleCursorAdapter オブジェクトを ListActivity に登録する

```
SimpleCursorAdapter simpleCursorAdapter = new SimpleCursorAdapter(  
    this,  
    android.R.layout.simple_list_item_1,  
    cursor,  
    new String[] { "name" },  
    new int[] { android.R.id.text1 },  
    CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);  
setListAdapter(simpleCursorAdapter);
```

10.12 【実習】データベース6



実習6 データの一覧表示

前回の実習で作成したプログラムを修正し、検索結果を一覧表示する。

手順

1. リソースファイルを修正する
2. 一覧画面を作成する
3. 一覧画面に検索結果の表示処理を追加する
4. MainActivity を修正する

手順1. リソースファイルを修正する

リソースファイルを次のように修正します。

strings.xml

表示するデータがなかったときのメッセージを追加します。

リスト 10.10: strings.xml

```
1: <string name="empty">データがありません</string>
```

検索結果画面のレイアウト

検索結果画面のレイアウトを次のように設定します。

ファイル名 : activity_result_list.xml

リスト 10.11: activity_result_list.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <ListView
6:         android:id="@android:id/list"
7:         android:layout_width="match_parent"
8:         android:layout_height="wrap_content" >
9:     </ListView>
10:
11:     <TextView
12:         android:id="@android:id/empty"
13:         android:layout_width="wrap_content"
14:         android:layout_height="wrap_content"
15:         android:text="@string/empty" />
16:
17: </LinearLayout>
```

手順 2. 一覧画面を作成する

ListActivity を継承した一覧画面の Activity を作成し、AndroidManifest.xml ファイルに登録します。

Cursor 型のメンバ変数を定義し、表示レイアウトファイルを「activity_result_list.xml」に設定します。

ファイル名 : ResultListActivity.java

```
public class ResultListActivity extends ListActivity {

    private Cursor cursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_result_list);
    }
}
```


手順 3. 一覧画面に検索結果の表示処理を追加する

ResultListActivity に検索処理を終了処理を追加します。

1. 検索処理を追加する

onStart メソッドをオーバーライドし、全件検索処理に追加します。

取得した Cursor オブジェクトを使って、SimpleCursorAdapter オブジェクトを生成します。setListAdapter メソッドを実行し、ResultListActivity に SimpleCursorAdapter オブジェクトを登録します。

```
protected void onStart() {
    super.onStart();
    SampleSQLiteOpenHelper helper = new SampleSQLiteOpenHelper(this);
    SQLiteDatabase database = helper.getReadableDatabase();

    // 全件検索する
    cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE,
        null, null, null, null, null, null);
    if (cursor != null) {
        SimpleCursorAdapter simpleCursorAdapter = new SimpleCursorAdapter(
            this,
            android.R.layout.simple_list_item_1,
            cursor,
            new String[] { "name" },
            new int[] { android.R.id.text1 },
            CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
        setListAdapter(simpleCursorAdapter);
    }
    helper.close();
}
```

2. Cursor をクローズする

onStop メソッドをオーバーライドし、Cursor をクローズします。

```
@Override
protected void onStop() {
    super.onStop();
    cursor.close();
}
```

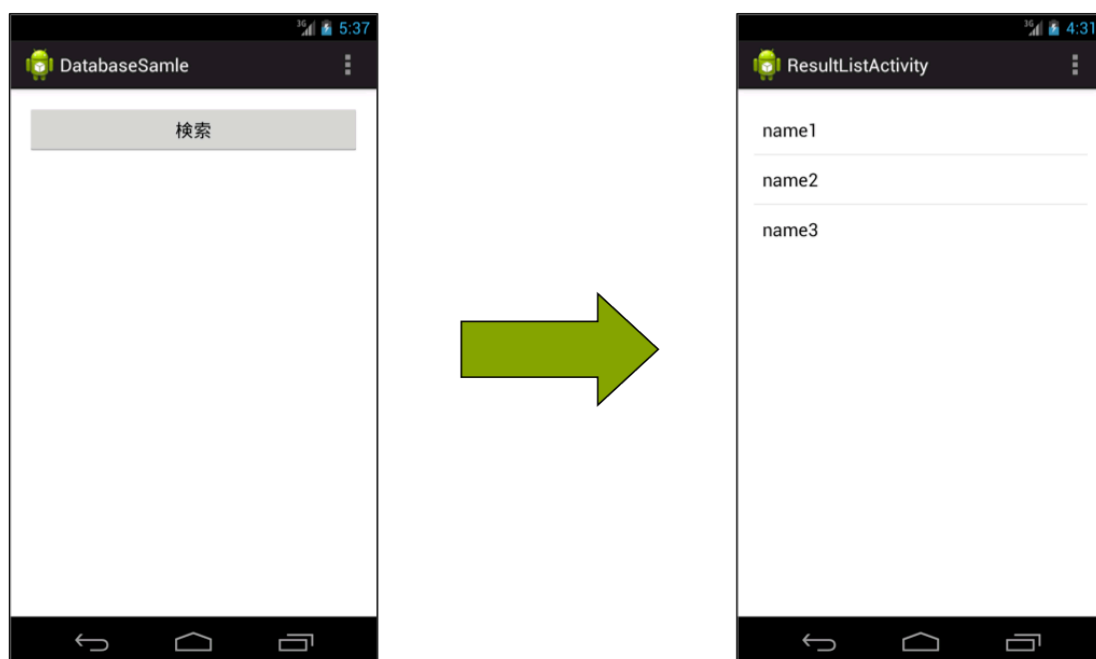
手順 4. MainActivity を修正する

検索画面からの遷移先を ResultListActivity に変更します。

```
public void onClickSearchButton(View v) {  
    Intent intent = new Intent(this, ResultListActivity.class);  
    startActivity(intent);  
}
```

確認

アプリケーションを実行し、検索結果が一覧表示されることを確認します。



解答

解答ドキュメントを参照してください。

10.13 条件検索

条件を指定して検索する方法

query メソッドの第 3 引数、第 4 引数に検索条件を指定することができます。

- 条件の指定方法
 - 方法 1 :
 - * 第 3 引数に where 句相当の文字列を記述する
 - 方法 2 :
 - * 第 3 引数に "[カラム名]=?" の書式で指定する
 - * 第 4 引数に "?" の値を保持した String 配列を指定する

方法 1 を使ったサンプル

第 3 引数に where 句相当の文字列を記述する方法です。ここでは、_id の値が 1 のデータを取得しています。

```
cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, "_id=1", null, null,
    null, null);
```

方法 2 を使ったサンプル

第 3 引数に "[カラム名]=?" の書式で指定する方法です。第 4 引数に "?" の値を保持した String 配列を指定します。ここでは、_id の値が 1 のデータを取得しています。

```
cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, "_id=?",
    new String[]{"1"}, null, null, null);
```

10.14 【実習】データベース7



データベース7

- 実習7
 - 前回の実習で作成したプログラムを修正し、詳細画面を表示する
 - 一覧画面から1件選択し、詳細画面でデータベースから条件検索を行う



① 検索ボタンを押してデータを取得



② 表示したいデータを選択する



③ 選択したデータが表示される

実習7 条件検索

前回の実習で作成したプログラムを修正し、詳細画面を表示する機能を追加します。

手順

1. リソースファイルを修正する
2. 詳細画面を追加する
3. 一覧画面に画面遷移処理を追加する
4. 詳細画面にデータを表示する処理を追加する

手順1. リソースファイルを修正する

リソースファイルを次のように修正します。

詳細画面のレイアウト

詳細画面のレイアウトを次のように設定します。

ファイル名 : activity_detail.xml

リスト 10.12: activity_detail.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView
6:         android:layout_width="wrap_content"
7:         android:layout_height="wrap_content"
8:         android:text="@string/name" >
9:     </TextView>
10:
11:     <TextView
12:         android:id="@+id/text_name"
13:         android:layout_width="wrap_content"
14:         android:layout_height="wrap_content"
15:         android:textAppearance="?android:attr/textAppearanceMedium" >
16:     </TextView>
17:
18:     <TextView
19:         android:layout_width="wrap_content"
20:         android:layout_height="wrap_content"
21:         android:text="@string/value" >
22:     </TextView>
23:
24:     <TextView
25:         android:id="@+id/text_value"
26:         android:layout_width="wrap_content"
27:         android:layout_height="wrap_content"
28:         android:textAppearance="?android:attr/textAppearanceMedium" >
29:     </TextView>
30:
31: </LinearLayout>
```

手順 2. 詳細画面を追加する

詳細画面の Activity を作成し、AndroidManifest.xml ファイルに登録します。

Cursor 型のメンバ変数を定義し、表示レイアウトファイルを「activity_detail.xml」に設定します。

ファイル名 : DetailActivity.java

```
public class DetailActivity extends Activity {
    private Cursor cursor;

    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detail);
}
```

手順 3. 一覧画面に画面遷移処理を追加する

ResultListActivity を修正し、アイテム選択時に詳細画面に遷移する処理を追加します。遷移するときに、選択アイテムの `_id` の値を Intent に格納します。

```
@Override
protected void onItemClick(ListView listView, View view, int position, long id) {
    Intent intent = new Intent(this, DetailActivity.class);
    intent.putExtra("id", id);
    startActivity(intent);
}
```

手順 4. 詳細画面にデータを表示する処理を追加する

DetailActivity に条件検索し、詳細データを表示する処理を追加します。

1. Intent から `_id` の値を取得する

DetailActivity の `onStart` メソッドで、Intent オブジェクトを取得します。Intent から選択アイテムの `_id` を取得します。

```
@Override
protected void onStart() {
    super.onStart();
    long id = getIntent().getLongExtra("id", -1);
}
```

2. 条件検索をし、データを表示する

取得した `_id` の値を使って、条件検索します。取得したデータを、TextView に設定します。

```
SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(this);
// 読込専用の SQLiteDatabase オブジェクトを取得する
SQLiteDatabase database = databaseOpenHelper.getReadableDatabase();
```

```
// 条件検索
cursor = database.query(SampleSQLiteOpenHelper.SAMPLE_TABLE, null, "_id=" + id, null,
    null, null, null);
if (cursor != null) {
    cursor.moveToFirst();

    // name をセット
    TextView textName = (TextView) findViewById(R.id.text_name);
    String name = cursor.getString(cursor.getColumnIndex("name"));
    textName.setText(name);

    // value をセット
    TextView textValue = (TextView) findViewById(R.id.text_value);
    String value = cursor.getString(cursor.getColumnIndex("value"));
    textValue.setText(value);
}
// データベースから切断する
databaseOpenHelper.close();
```

3. Cursor をクローズする

onStop メソッドをオーバーライドし、Cursor をクローズします。

```
@Override
protected void onStop() {
    super.onStop();
    cursor.close();
}
```

確認

検索結果一覧画面から 1 件選択し、選択したデータが詳細画面で表示されることを確認します。



解答

解答ドキュメントを参照してください。

10.15 データの更新

データを更新する方法

データベースのデータを更新するためには、SQLiteDatabase の update メソッドを使用します。

```
public int update(String table,
                  ContentValues values,
                  String whereClause,
                  String[] whereArgs)
```

引数には更新したいレコード情報を指定します。戻り値は更新件数です。更新に失敗した場合は、-1 が戻り値として返却されます。

update メソッドのパラメータは次のようになります。

String table:

テーブル名

ContentValues values:

更新データのカラム名と値をもった ContentValues オブジェクト

String whereClause:

WHERE 句相当の条件式

String[] whereArgs:

第 3 引数に "?" が含まれる場合に置き変わる値を保持した String 配列

リスト 10.13: update の例

```
1: SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(
2:     this);
3: SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();
4:
5: // update データの設定
6: ContentValues values = new ContentValues();
7: values.put("name", name);
8: values.put("value", value);
9:
10: // データを更新する
11: long result = database.update(SampleSQLiteOpenHelper.SAMPLE_TABLE,
12:     values, "_id=" + id, null);
13:
14: databaseOpenHelper.close();
15: if (result != -1) {
16:     Toast.makeText(this, "更新完了", Toast.LENGTH_LONG).show();
17:     finish();
18: }
```


10.16 【実習】データベース 8



 データベース 8

- 実習 8
 - データの更新処理
 - 前回の実習で作成したプログラムを修正し、データの更新機能を追加する

実習 8 データの更新

前回の実習で作成したプログラムを修正し、データの更新機能を追加します。

手順

1. リソースファイルを修正
2. 更新画面を作成
3. 詳細画面からの画面遷移処理を追加
4. データベース更新処理を追加

手順 1. リソースファイルを修正

strings.xml

次の文字列リソースを追加します。

リスト 10.14: strings.xml

```
1: <string name="edit">修正</string>
2: <string name="update">更新</string>
3: <string name="update_complete">更新完了</string>
```

詳細画面のレイアウト

詳細画面のレイアウトを次のように設定します。

リスト 10.15: activity_detail.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView ... />
6:     <TextView ... />
7:     <TextView ... />
8:     <TextView ... />
9:
10:    <Button
11:        android:id="@+id/button_edit"
12:        android:layout_width="match_parent"
13:        android:layout_height="wrap_content"
14:        android:onClick="onClickEditButton"
15:        android:text="@string/edit" >
16:    </Button>
17:
18: </LinearLayout>
```

更新画面のレイアウト

更新画面のレイアウトを次のように設定します。

リスト 10.16: activity_update.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView
6:         android:layout_width="wrap_content"
7:         android:layout_height="wrap_content"
8:         android:text="@string/name" >
9:     </TextView>
10:
11:    <EditText
12:        android:id="@+id/edit_name"
13:        android:layout_width="match_parent"
14:        android:layout_height="wrap_content" >
```

```
15:     </EditText>
16:
17:     <TextView
18:         android:layout_width="wrap_content"
19:         android:layout_height="wrap_content"
20:         android:text="@string/value" >
21:     </TextView>
22:
23:     <EditText
24:         android:id="@+id/edit_value"
25:         android:layout_width="match_parent"
26:         android:layout_height="wrap_content"
27:         android:inputType="number" >
28:     </EditText>
29:
30:     <Button
31:         android:id="@+id/button_decide"
32:         android:layout_width="match_parent"
33:         android:layout_height="wrap_content"
34:         android:onClick="onClickUpdateButton"
35:         android:text="@string/update" >
36:     </Button>
37:
38: </LinearLayout>
```

手順 2. 更新画面を作成

更新画面の Activity を作成し、AndroidManifest.xml ファイルに登録します。

onCreate メソッドでメンバ変数の初期化を行い、更新対象の現在の設定値を表示します。ファイル名: UpdateActivity.java

```
public class UpdateActivity extends Activity {

    private EditText editName;
    private EditText editValue;
    private long id;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_update);

        editName = (EditText) findViewById(R.id.edit_name);
        editValue = (EditText) findViewById(R.id.edit_value);

        // Intent から値を取得
        Intent intent = getIntent();
        id = intent.getLongExtra("id", -1);
        String name = intent.getStringExtra("name");
        String value = intent.getStringExtra("value");
```

```
// name をセット
editName.setText(name);
// value をセット
editValue.setText(value);
}
```

手順 3. 詳細画面からの画面遷移処理を追加

DetailActivity を修正し、更新ボタンクリック時に更新画面に遷移する処理を追加します。遷移するときに、選択アイテムの `_id`、`name`、`value` の値を Intent に格納します。

```
public void onClickEditButton(View v) {
    Bundle extras = getIntent().getExtras();
    long id = extras.getLong("id");

    TextView textName = (TextView) findViewById(R.id.text_name);
    TextView textValue = (TextView) findViewById(R.id.text_value);
    Intent intent = new Intent(this, UpdateActivity.class);
    intent.putExtra("id", id);
    intent.putExtra("name", textName.getText().toString());
    intent.putExtra("value", textValue.getText().toString());
    startActivity(intent);
}
```

手順 4. データベース更新処理を追加

更新ボタンクリック時に、EditText に入力された値をデータベースに登録する処理を追加します。

```
public void onClickUpdateButton(View v) {
    String name = editName.getText().toString();
    String value = editValue.getText().toString();

    if (!TextUtils.isEmpty(name) && !TextUtils.isEmpty(value) ){
        SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(
            this);
        SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();

        // update データの設定
        ContentValues values = new ContentValues();
        values.put("name", name);
        values.put("value", value);

        // データを更新する
        long result = database.update(SampleSQLiteOpenHelper.SAMPLE_TABLE,
            values, "_id=" + id, null);
    }
}
```

```
databaseOpenHelper.close();
if (result != -1) {
    Toast.makeText(this, R.string.update_complete,
        Toast.LENGTH_LONG).show();
    finish();
}
}
```

確認

アプリケーションを実行し、データが更新されていることを確認します。



解答

解答ドキュメントを参照してください。

10.17 データの削除

データを削除する方法

データベースのデータを削除するためには、`SQLiteDatabase` の `delete` メソッドを使用します。データの削除に失敗した場合は-1 削除できた場合はレコード数が戻り値として返却されます。

```
public int delete(String table, String whereClause, String[] whereArgs)
```

引数には削除したいレコード情報を指定します。戻り値は削除件数です。削除に失敗した場合は、-1 が戻り値として返却されます。

`delete` メソッドのパラメータは次のようになります。

String table:

テーブル名

String whereClause:

WHERE 句相当の条件式

String[] whereArgs:

第3引数に "?" が含まれる場合に置き変わる値を保持した String 配列

リスト 10.17: `delete` の例

```
1: SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(this);
2: SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();
3:
4: // データを削除する
5: long result = database.delete(SampleSQLiteOpenHelper.SAMPLE_TABLE, "_id=" + id,
6:     null);
7: // データベースから切断する
8: databaseOpenHelper.close();
9: if (result != -1) {
10:     Toast.makeText(this, "削除完了", Toast.LENGTH_LONG).show();
11:     finish();
12: }
```


10.18 【実習】データベース 9



- 実習 9
 - データの削除処理
 - サンプルプログラムを修正してデータを削除する機能を追加する

実習 8 データの削除

前回の実習で作成したプログラムを修正し、データの削除機能を追加します。

手順

1. リソースファイルを修正
2. DetailActivity の修正

手順 1. リソースファイルを修正

strings.xml

次の文字列リソースを追加します。

```
<string name="delete_complete">削除完了</string>
```

詳細画面のレイアウト

詳細画面のレイアウトを次のように設定します。

リスト 10.18: activity_main.xml

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     ...
3:     >
4:
5:     <TextView ... />
6:     <TextView ... />
7:     <TextView ... />
8:     <TextView ... />
9:     <Button ... />
10:
11:     <Button
12:         android:id="@+id/button_delete"
13:         android:layout_width="match_parent"
14:         android:layout_height="wrap_content"
15:         android:onClick="onClickDeleteButton"
16:         android:text="@string/delete" >
17:     </Button>
18:
19: </LinearLayout>
```

手順 2. DetailActivity の修正

DetailActivity を修正し、削除ボタンクリック時に選択アイテムの削除処理を追加します。

```
public void onClickDeleteButton(View v) {
    long id = getIntent().getLongExtra("id", -1);

    SampleSQLiteOpenHelper databaseOpenHelper = new SampleSQLiteOpenHelper(this);
    SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();

    // データを削除する
    long result = database.delete(SampleSQLiteOpenHelper.SAMPLE_TABLE, "_id=" + id, null);
    // データベースから切断する
    databaseOpenHelper.close();
    if (result != -1) {
        Toast.makeText(this, R.string.delete_complete, Toast.LENGTH_LONG).show();
        finish();
    }
}
```

確認

アプリケーションを実行し、データが削除されていることを確認します。



解答

解答ドキュメントを参照してください。

平成 25 年度「成長分野等における中核的専門人材養成の戦略的推進」事業
スマホアプリ開発技術者育成のためのカリキュラム・教材開発と評価指標検証

Android アプリケーション開発教材

平成 26 年 2 月

学校法人電子学園 日本電子専門学校
〒169-8522 東京都新宿区百人町 1-25-4
Tel : 03-3369-9333

●本書の内容を無断で転記、掲載することは禁じます。