2024

2024 年マイコン部講習会
Android アプリコース

第 2 回講習会資料

講師　大政悠暉

目次

8 章 Java の基本文法

前節では Android Studio のデザインエディタを使用して、自己紹介アプリの部品を配置しました。今回は Java の部品の処理を記述するために、まずは Java の基本文法を学びます。

8.1 Java の基本説明

まずはこれから学ぶ Java について知りましょう。Java は、プログラミング言語の一つで、世界中で広く使われています。この言語は、ウェブアプリケーション、モバイルアプリケーション、デスクトップアプリケーションなど、さまざまな種類のソフトウェアを開発するために使用されます。Java は、その堅牢性、セキュリティ、移植性により、多くの開発者から高く評価されています。

8.1.1 Java とは何か

Java は、1995 年にサン・マイクロシステムズ（現オラクル）によって開発された、オブジェクト指向のプログラミング言語です。世界中で広く使われており、デスクトップアプリケーションから Web アプリケーション、モバイルアプリケーション（特に Android アプリ）、さらにはエンタープライズシステムまで、さまざまな分野で利用されています。

8.1.2 なぜ Java を学ぶのか？

Java を学ぶことには多くの利点があります。まず、Java は非常に人気があり、多くの企業で使用されているため、Java を習得することで就職やキャリアアップの機会が広がります。また、Java は多くの大規模システムで使用されているため、堅牢なプログラムを作成するスキルを身につけることができます。

1. **普及度と需要**

   Java は非常に広く普及しているため、多くの企業が Java を使用しています。そのため、Java のスキルを持っていると就職やキャリアアップに有利です。

2. **プラットフォームの独立性**

   Java で書かれたプログラムは、Java Virtual Machine（JVM）という仕組みを使って実行されます。これにより、Windows、Mac、Linux など、異なるプラットフォームでも同じコードが動作します。

3. **豊富なライブラリとフレームワーク**

   Java には、多くのライブラリとフレームワークが用意されており、開発を効率的に進めることができます。例えば、Web アプリケーション開発には Spring、データベース操作には Hibernate などがあります。

4. **Android アプリ開発**

   Android アプリの開発には、Java が主要なプログラミング言語として使われています。Android アプリの開発を目指すなら、Java の知識は必須です。

8.1.3 Java の特徴

プログラミング言語には、それぞれ作られたときのコンセプトや考え方に応じて特徴があります。Java 言語には以下のような特徴があります。

1. **オブジェクト指向**

   Java はオブジェクト指向言語であり、データとその操作を一つにまとめた「オブジェクト」を中心にプログラムを構築します。これにより、コードの再利用性や保守性が向上します。

2. **安全性**

   Java はメモリ管理を自動化しており、プログラムの安全性を高めるための機能が充実しています。例えば、ポインタを使用しないことでメモリリークやセキュリティホールのリスクを減らしています。

3. **マルチスレッド**

   Java はマルチスレッドプログラミングをサポートしており、複数の作業を同時に実行することができます。これにより、効率的なプログラムを作成することが可能です。

8.1.4 Java プログラムの基本構成

Java プログラムは、クラスの定義から始まり、クラスの中にフィールド（データ）とメソッド（操作）が含まれます。以下に、簡単な Java プログラムの例を示します。Java プログラムは、次のような基本構成を持っています。

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

1. **クラス（class）**

   Java プログラムは、クラスという単位で構成されます。上記の例では、「Main」というクラスが定義されています。

2. **メソッド（method）**

   メソッドは、クラス内に定義される一連の処理です。上記の例では、「main」というメソッドが定義されています。この「main」メソッドは、プログラムのエントリーポイントであり、プログラムが最初に実行される場所です。

3. **処理の内容**

   メソッドの中には、処理の内容が記述されます。今回は、「Hello, World!」と出力する処理が記述されています。

   System.out.println("Hello, World!");は、コンソールに文字列を出力するためのコードです。このようにして、プログラムの実行結果を表示します。（コンソールで実行したときに限りこの方法を使用します。Android Studio では後ほど紹介する Toast という方法を使用します）

コラム：Java の実行

これから実行するプログラムは Java 言語の汎用的な文法のため、一部 Android Studio では動かないものもあります。したがって、以下の URL からインターネット上で Java を実行できるサイトにアクセスし、そこでプログラムを実行しましょう。

オンライン Java コンパイラ

URL：https://www.mycompiler.io/ja/online-java-compiler

8.2 変数とデータ型

プログラミングにおいて、変数はデータを格納するための名前付きの記憶領域です。データ型は、変数に格納できるデータの種類を定義します。Java には、さまざまなデータ型があり、それぞれ異なる特性を持っています。

8.2.1 変数とは

変数とは、データを一時的に保存するための「箱」のようなものです。変数に値を入れておくことで、その値を後で利用することができます。変数を使うことで、プログラムが柔軟に動作するようになります。

8.2.2 変数の宣言と初期化

変数を使うには、まず変数を「宣言」し、その後に「初期化」する必要があります。

宣言　　：変数の名前とデータの種類を指定すること

初期化　　：変数に最初の値を与えること

例えば以下のように宣言、初期化することができます。

```
int age = 20; // 整数型の変数ageを宣言し、20で初期化
String name = "Taro"; // 文字列型の変数nameを宣言し、「Taro」で初期化
```

8.2.3 基本データ型

データ型は、変数に格納されるデータの種類を指定します。Java には、基本データ型（プリミティブ型）と参照型の 2 種類があります。

基本データ型　：数値、文字、ブール値などが含まれます

参照型　　：文字列や配列、クラスが含まれます

以下に主要な型を紹介します。

整数型（int, long）

基本データ型で。整数値を扱うためのデータ型です。

```
int number = 5; // 整数型の変数numberを宣言し、5で初期化
long bigNumber = 123456789L; // 長い整数型の変数bigNumberを宣言し、123456789で初期化
```

小数型（float, double）

基本データ型で、小数点を含む数値を扱うためのデータ型です。

```
float height = 175.5f; // 小数型の変数heightを宣言し、175.5で初期化
double weight = 65.3; // 小数型の変数weightを宣言し、65.3で初期化
```

文字型（char）

基本データ型で、一つの文字を扱うためのデータ型です。

```
char initial = 'A'; // 文字型の変数initialを宣言し、'A'で初期化
```

文字列型（String）

参照データ型で、文字の連なり（文字列）を扱うためのデータ型です。

```
String message = "Hello, World!"; // 文字列型の変数messageを宣言し、「Hello, World!」
```

論理型（boolean）

基本データ型で、真（true）または偽（false）を扱うためのデータ型です。

```
boolean isStudent = true; // 論理型の変数isStudentを宣言し、trueで初期化
```

8.2.4 型変換

Java では、異なるデータ型間で値を変換することができます。これを型変換と呼びます。型変換には、暗黙的型変換（自動的に行われる）と明示的型変換（直接プログラムを書いて型変換（キャスト）が必要）の 2 種類があります。

暗黙的型変換の例

```
int num = 10;
double result = num; // int型からdouble型への暗黙的型変換
```

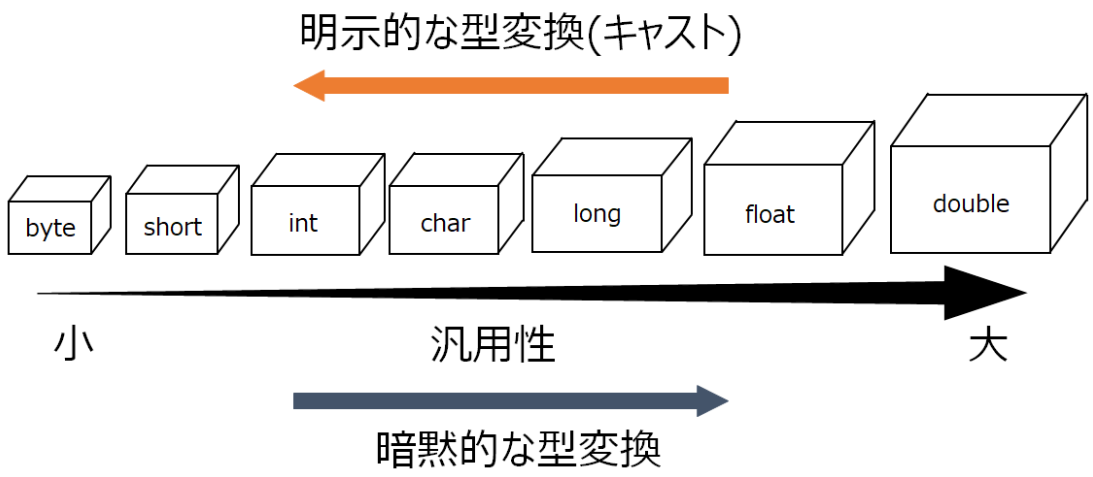
上記のプログラムでは、整数型の変数 int を小数型の変数 result に代入しています。この時特に何も書かなくても自動的に型変換が行われ、result には 10.000…が代入されます。

明示的型変換の例

```
double value = 9.99;
int intValue = (int) value; // double型からint型への明示的型変換
```

上記のプログラムでは、小数型の変数 value を整数型の変数 intValue に代入しています。型変換を行う際は、変換先の型を()で囲って指定します。このように、より適用範囲が広い型から狭い型に型変換を行うときは、型変換を明示する必要があります。

型の適用範囲とは、その方に入力することができる文字や数字の種類の事です。例えば int 型は 1,2,3…といった整数しか扱えないのに対し、小数型の float 型は、1.00…という章数も扱うことができます。この時、float 型は int 型よりも適用範囲が広く、汎用性が高いということになります。



8.2.5 まとめ

変数とデータ型は、プログラミングにおいて非常に重要な概念です。これを理解することで、データを効率的に扱うことができるようになります。余裕があれば実際に変数を使って実際にコードを書いてみましょう。

8.3 条件分岐

プログラムにおいて、特定の条件に応じて異なる処理を行うことができるのは非常に重要です。たとえば、ユーザーが入力したデータが正しいかどうかをチェックしたり、ゲームでのスコアによって異なるメッセージを表示したりする場合など、条件分岐はさまざまな場面で使われます。これにより、プログラムが柔軟に対応できるようになります。

Java では、条件分岐を実現するためにいくつかの構文を提供しています。ここでは、基本的な条件分岐構文である if 文と else if 文、複数の条件を組み合わせる方法について説明します。これらの構文を理解することで、プログラムの処理をよりきめ細かく制御できるようになります。

8.3.1 if 文の基本

if 文は、特定の条件が満たされているかどうかをチェックし、その条件が真（true）であれば、指定された処理を実行します。条件が偽（false）であれば、処理はスキップされます。プログラムの例は以下のようになります。

```
int score = 85;
if (score >= 90) {
    System.out.println("Excellent!");
} else if (score >= 70) {
    System.out.println("Good Job!");
} else {
    System.out.println("Keep Trying!");
}
```

この例では、スコアが 90 以上の場合は「Excellent!」、70 以上 90 未満の場合は「Good Job!」、それ以外の場合は「Keep Trying!」と表示されます。

また、複数の条件をチェックしたい場合、**else if** を使います。**else if** は、前の条件が満たされなかった場合に、次の条件をチェックするために使用されます。これにより、プログラムが複数の状況に対応できるようになります。

8.3.2 複数条件の処理

Java では、複数の条件を組み合わせて使うこともできます。これには論理演算子（&&、||）を使用します。

- && (AND)

  両方の条件が真である場合に成り立ちます。

```
int age = 25;
boolean isStudent = true;
if (age >= 18 && isStudent) {
    System.out.println("You are an adult student.");
}
```

  今回は年齢 age が 18 以上かつブーリアン型 isStudent が true だった場合、条件が満たされて「You are an adult student」と表示されます。

- || (OR)

  どちらか一方の条件が真であれば成り立ちます。

```
boolean hasPassport = false;
boolean hasVisa = true;
if (hasPassport || hasVisa) {
    System.out.println("You can travel.");
}
```

  今回はブーリアン型 hasPassport と hasVisa のどちらかが true だったら場合、条件が満たされて「You can travel.」と表示されます。

条件分岐をマスターすることで、プログラムの流れを柔軟に制御できるようになります。
次に、繰り返し処理について学びましょう。

8.4 繰り返し処理

プログラムには、同じ処理を何度も繰り返す必要がある場合があります。これを繰り返し処理と呼びます。繰り返し処理を使うことで、コードを短くシンプルに保つことができ、効率的なプログラムを書くことができます。

Java では、繰り返し処理を実現するために、主に for 文と while 文を使用します。ここでは、これらの基本的な使い方について説明します。繰り返し処理を理解することで、ループを効果的に使い、複雑なタスクを簡単に処理することができるようになります。

8.4.1 for 文の基本

for 文は、指定した回数だけ繰り返すための構文です。ループの開始時点、終了条件、繰り返しごとの処理を一つの行で指定します。プログラムの例は以下のようになります。

```
for (int i = 0; i < 5; i++) {
    System.out.println("Count: " + i);
}
```

この例では、変数 **i** が 0 から始まり、5 未満の間、つまり 0 から 4 まで繰り返し処理が行われます。**i** の値が 1 ずつ増加しながら、ループのたびに「Count: i」というメッセージが表示されます。

8.4.2 while 文の基本

while 文は、条件が真である限り繰り返すための構文です。条件が偽（false）になるまで繰り返します。

```
int count = 0;
while (count < 5) {
    System.out.println("Count: " + count);
    count++;
}
```

この例では、**count** の値が 0 から始まり、5 未満の間、ループが繰り返されます。**count** の値が 1 ずつ増加しながら、ループのたびに「Count: count」というメッセージが表示されます。

これらを理解することで、プログラムの柔軟性と効率性が大幅に向上します。実際に手を動かして、これらの構文を使ったコードを書いてみましょう。

8.5 メソッドの定義と呼び出し

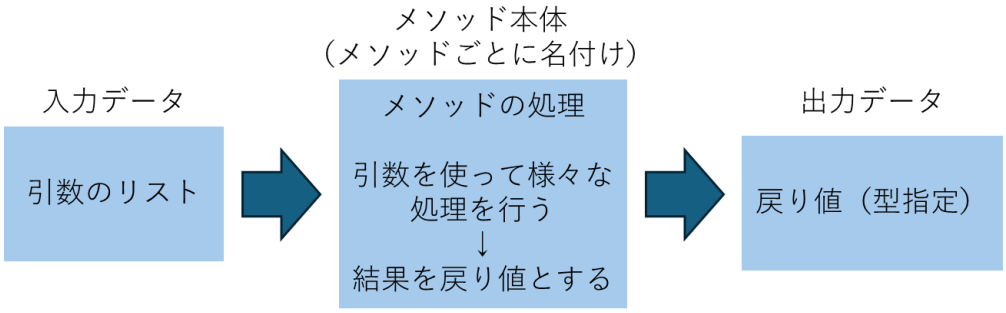
プログラムを作成する際、同じ処理を何度も書くのは効率的ではありません。そこで、処理をまとめて再利用できるようにするために「メソッド」を使います。メソッドは、特定の機能を持つコードのブロックで、何度でも呼び出して使用することができます。これにより、コードが整理され、読みやすくなります。

メソッドを使うことで、プログラムを小さな部分に分割して管理することができます。これにより、プログラム全体の構造が明確になり、デバッグや修正が容易になります。特に大規模なプログラムでは、メソッドの利用は不可欠です。

なお、似たような役割を持つものに関数がありますが、メソッドと関数の違いはほぼなく、後述するクラスに属していればメソッド、属していなければ関数と呼ばれます。

8.5.1 メソッド(関数)の定義

メソッドを定義する際には、メソッド名、引数（受け取るデータ）リスト、戻り値(返却するデータ)の型、そしてメソッドの本体を指定します。これを図に表すと以下のようになります。



これを踏まえて、基本的な構文は以下の通りです。下の例はクラスに属していないため、正しくは関数の定義になります。

```
public static 戻り値の型 メソッド名(引数リスト) {
    // メソッドの処理内容
}
```

例えば、2 つの整数を足し合わせてその結果を返すメソッドを定義する場合は、次のようになります。

```java
public static int add(int a, int b) {
    int sum = a + b;
    return sum;
}
```

この例では、**add** というメソッドを定義し、2 つの整数 **a** と **b** を引数として受け取り、その合計 **sum** を戻り値として返します。

8.5.2 メソッド(関数)の呼び出し

メソッドを定義した後は、必要なときにそのメソッドを呼び出して利用することができます。メソッドの呼び出しは、メソッド名と引数を指定して行います。この例も、元の関数 add がクラスに属していないため、関数の呼び出しの例となります。

```java
int result = add(5, 3);
System.out.println("Result: " + result);
```

この例では、8.5.1 の **add** メソッドを呼び出し、5 と 3 を引数として渡し、その結果を **result** に格納します。その後、結果を画面に表示します。したがって、今回は 8 が出力されます。

8.5.3 メソッドのまとめ

メソッドを利用することで、コードを再利用可能にし、プログラム全体を整理することができます。これにより、プログラムのメンテナンスが容易になり、バグの発生を減らすことができます。次に、Java の重要な概念である「クラス」と「オブジェクト」について学びましょう。

8.6 クラスとオブジェクト

プログラミングにおいて、「クラス」と「オブジェクト」は非常に重要な概念です。これらは、オブジェクト指向プログラミング（OOP）の基礎であり、Java を理解するためには欠かせない要素です。オブジェクト指向プログラミングは、現実世界の物事をプログラムの中でモデル化する方法を提供します。

オブジェクト指向は、初心者には少し難しい概念かもしれませんが、例を交えながら分かりやすく説明します。

オブジェクト指向の説明

オブジェクト指向の基本的な考え方は、機能を部品化し、部品化された機能を組み合わせることで製品を完成させる考え方です。

オブジェクト指向プログラミングでは、オブジェクトという「部品」を組み合わせてプログラムを構築します。各オブジェクトは、各オブジェクトは、自分のデータ（例えば車の燃料の量など）と機能（アクセルを踏むなどの操作方法）を持ち、その機能やデータに対する操作（メソッド）を外部から利用できます。これらを組み合わせて、複雑なプログラムを簡単に作成できます。オブジェクトはそれぞれが自立していて、お互いに影響を与えることなく動作します。したがって、オブジェクト指向により複雑なシステムを理解しやすく、再利用しやすい部品に分割することができます。

オブジェクト指向を導入することのメリットとして、現実世界で私たちは車の詳細な構造を知らなくても、車を運転することはできます。これは、アクセルを踏んだりハンドルを動かしたりする（入力）と車が動いて曲がる（出力）さえわかっていれば、内部の複雑な仕組み（エンジンの動作や電気系統など）を知らなくても車を使用できるということです。車の内部がどのように動いているかは、運転者にとっては「ブラックボックス」として扱われます。

さらに、車の種類が変わったとしても、例えば車体の色やエンジンの種類、内装などが変わっても、運転の基本操作は変わりません。同じように、プログラミングにおいても、オブジェクト指向プログラミングでは、プログラムの内部処理やデータの詳細を知らなくても、オブジェクトに対して値を入力すれば、期待する出力を得ることができます。

つまり、オブジェクト指向プログラミングは、現実世界のオブジェクト（車など）の操作方法と似ており、各オブジェクトが自分自身のデータとメソッドを持ち、それを外部から操作することができるプログラミング手法です。

8.6.1 クラスとは何か

クラスは、オブジェクト指向プログラミングにおいて重要な概念です。クラスは、データ（フィールド）と操作（メソッド）をひとまとめにした設計図のようなものです。クラスを使うことで、プログラムの中でオブジェクトを生成し、そのオブジェクトが持つ特性や振る舞いを定義することができます。プログラムでクラスを使用したいときは、以下のように宣言することができます。

```
public class ClassName {
    // フィールド（属性）
    dataType fieldName;

    // コンストラクタ（オブジェクトの初期化）
    public ClassName(parameters) {
        // 初期化処理
    }

    // メソッド（操作）
    public returnType methodName(parameters) {
        // メソッドの処理内容
    }
}
```

例えば、「犬」というクラスを考えてみましょう。犬には名前や年齢といった属性（フィールド）があり、「吠える」や「走る」といった動作（メソッド）を持っています。すると、以下のようにすることで簡単なクラスを宣言することができます。

```java
public class Dog {
    // フィールド（属性）
    String name;
    int age;

    // コンストラクタ（オブジェクトの初期化）
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // メソッド（動作）
    public void bark() {
        System.out.println(name + " is barking!");
    }

    public void run() {
        System.out.println(name + " is running!");
    }
}
```

フィールドは、クラス内で定義される変数であり、オブジェクトの状態を表します。上記の例では、name と age がフィールドです。これらのフィールドは、クラス内のどのメソッドからもアクセスできます。

コンストラクタは、オブジェクトが作られる時に呼び出される特別なメソッドで、オブジェクトの初期化を行います。コンストラクタの名前はクラス名と同じで、戻り値はありません。コンストラクタ内にある this は、現在のオブジェクトを指すキーワードです。コンストラクタ内でフィールドを初期化する際、パラメータ名とフィールド名が同じ場合に、this を使ってフィールドを明確に区別します。

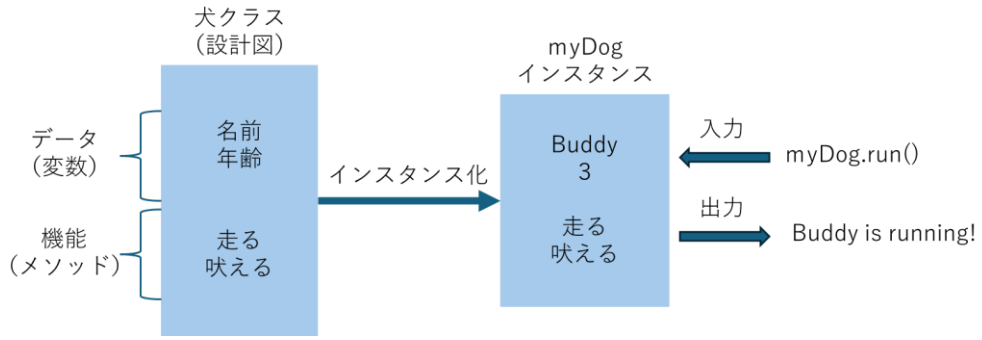
8.6.2 オブジェクトとは何か

オブジェクトは、クラスから作られた実体です。クラスが設計図であるのに対し、オブジェクトはその設計図から作られた製品のようなものです。オブジェクトは、クラスで定義されたフィールドとメソッドを持っています。

```java
public class Main {
    public static void main(String[] args) {
        // Dogクラスからオブジェクトを作成
        Dog myDog = new Dog("Buddy", 3);

        // オブジェクトのメソッドを呼び出し
        myDog.bark();
        myDog.run();
    }
}
```

この例では、Dog クラスから myDog というオブジェクト（インスタンス）を作成し、そのオブジェクトの bark メソッドと run メソッドを呼び出しています



8.6.3 オブジェクト指向の特徴

オブジェクト指向プログラミングには、いくつかの重要な特徴があります。ここでは、最も基本的な特徴を紹介します。

- **カプセル化**

  データ（フィールド）とメソッドをクラスにまとめることで、データの保護と管理を容易にします。これは先ほどの説明の入力と出力がわかっていたら中身は知らなくても大丈夫というものです。

- **継承**

  既存のクラスを基に新しいクラスを作成し、コードの再利用を促進します。これはもともとのクラスとほとんど同じだが若干異なる処理がある場合、継承を行ってクラス

を使いまわし、異なる部分の処理だけ書き直すといったことができます。これは、先ほどの説明の部品の再利用に関係しています。

- **ポリモーフィズム**

  同じメソッド名でも、異なるクラスで異なる動作を実現できます。

オブジェクト指向プログラミングの理解は、Java だけでなく、多くの現代のプログラミング言語で共通しています。これにより、より多くのプログラマが同じコードベースで協力しやすく、これらの特徴を理解することで、より柔軟で再利用可能なコードを書くことができます。

8.6.4 まとめ

クラスとオブジェクト、そしてオブジェクト指向プログラミングの基本概念を理解することは、Java プログラミングを学ぶ上で非常に重要です。クラスとオブジェクトを使って、現実世界の物事をモデル化し、プログラムの中で扱うことができます。オブジェクト指向の特徴を活かして、効率的で拡張性のあるプログラムを作成しましょう。次に、これまで学んだことを実践し、簡単なプロジェクトを通じて理解を深めていきます。

9 章　自己紹介アプリの処理部分の実装

ここまでで、Java の基本文法を学びました。次は、実際に自己紹介アプリに機能を追加していきましょう。前回の講習では、自己紹介アプリのデザイン部分（ボタンや入力ボックスの配置など）を完了しました。今回は、このアプリに対して入力を受け取り、適切なメッセージを表示する処理を実装します。

9.1 ボタンのクリックイベントの実装

まずは、ボタンがクリックされたときに何をするかを決めます。具体的には、入力ボックスに入力された名前を取得し、それを使ってメッセージを表示します。この節では、ボタンのクリックイベントの設定方法と、EditText からのテキスト取得、TextView へのテキスト設定について学びます。

9.1.1　プログラム実装

実装の流れ

1. **EditText と Button の参照を取得する**

   ボタンのクリックイベントを設定する前に、レイアウトに配置した EditText と Button をコード内で参照する必要があります。

2. **ボタンのクリックイベントを設定する**

   ボタンがクリックされたときに実行する処理を記述します。

3. **入力された名前を取得し、メッセージを表示する**

   EditText から名前を取得し、その名前を使って TextView にメッセージを表示します。また、Toast を使ってメッセージを表示します。

プログラム

対応する java のプログラムファイルを開いて、次のように入力してください。

```java
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity3 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main3);

        //==========ここから入力==========
        // レイアウト内のEditTextとButtonの参照を取得
        EditText nameEditText = findViewById(R.id.editTextText2);
        Button greetButton = findViewById(R.id.button2);
        TextView greetingTextView = findViewById(R.id.textView4);

        // ボタンのクリックイベントを設定
        greetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 入力された名前を取得
                String name = nameEditText.getText().toString();

                // メッセージを表示
                greetingTextView.setText(name + "さん、こんにちは！");

                // Toastでメッセージを表示
                Toast.makeText( context: MainActivity3.this,  text: name + "さん、こんにちは！", Toast.LENGTH_SHORT).show();
            }
        });
        //==========ここまで入力==========
    }
}
```

なお、上の画像のアクティビティ名やビューの ID は各自変更してください。

また、そのままプログラムをコピペすると赤文字になる箇所や赤色の波線が出る場所があると思います。その箇所はエラーとなっており、そのままではプログラムを実行することはできません。たいていの場合、エラーは必要な機能を import できていないことが原因です。Android Studio は自動で必要な機能をインポートしてくれますが、自動で入力するには対応している関数の一部を記述する必要があります。したがって、赤色の単語の最後の文字を消して再度打ち込み、入力補助の予測リストが出てきたら TAB キーを押して確定してください。赤文字および赤波線がすべてなくなれば OK です。

9.1.2 コードの解説

先ほどのプログラムでは、いくつか Android Studio 独自の命令が使用されています。ここでは、それらの命令を解説します。

- findViewById(R.id.nameEditText)

  XML レイアウトファイルで定義された EditText の ID を使って、対応するビューオブジェクトを取得します。つまり、java のファイル上で使いたい対象の XML のビューを指定する際にこの命令を使用します。

- greetButton.setOnClickListener(new View.OnClickListener() {...})

  ボタンにクリックリスナーを設定し、ボタンがクリックされたときに実行する処理を記述します。

- nameEditText.getText().toString()

  EditText に入力されたテキストを取得し、文字列として保存します。

- Toast.makeText(MainActivity.this, ...).show()

  簡単なメッセージを短時間表示するための Toast を作成して表示します。

9.1.3 デバッグのポイント

実際にプログラムを作成する際は、ほぼ確実といってよいほどバグ（プログラム上の不具合）が発生します。そのバグを特定して修正するのがデバッグです。今回のデバッグのポイントは以下のようになります。

- 実際にボタンをクリックして、メッセージが表示されるか確認しましょう。
- 名前が取得されない場合、EditText の ID が間違っていないか確認しましょう。
- アプリがクラッシュする場合、Logcat を使ってエラーメッセージを確認し、問題を特定しましょう。

9.2 テーマの変更

次に、画面下部に配置したスイッチを使ってアプリのライトテーマとダークテーマを切り替える機能を実装します。テーマの切り替えは、アプリのユーザーエクスペリエンスを向上させる重要な機能です。

9.2.1 プログラム実装

実装の流れ

1. **スイッチの参照を取得する**

   スイッチの参照を取得します。

2. **スイッチの状態変更イベントを設定する**

   スイッチの状態が変更されたときにテーマを切り替える処理を記述します。

プログラム

9.1 で作成したプログラムに処理を追加し、以下のようにしてください。

```
        @Override
        public void onClick(View v) {
            // 入力された名前を取得
            String name = nameEditText.getText().toString();

            // メッセージを表示
            greetingTextView.setText(name + "さん、こんにちは！");

            // Toastでメッセージを表示
            Toast.makeText( context: MainActivity3.this,  text: name + "さん、こんにちは！", Toast.LENGTH_SHORT).show();
        }
    });
    //=====================ここから入力=====================
    // スイッチの参照を取得
    Switch themeSwitch = findViewById(R.id.switch2);
    // スイッチの状態変更イベントを設定
    themeSwitch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked) {
                // ダークテーマに切り替え
                AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);
            } else {
                // ライトテーマに切り替え
                AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
            }

            // Toastでテーマの変更を通知
            String theme = isChecked ? "ダークテーマ" : "ライトテーマ";
            Toast.makeText( context: MainActivity3.this,  text: theme + "に変更されました", Toast.LENGTH_SHORT).show();
        }
    });
    //=====================ここまで入力=====================
}
}
```

今回も前節と同じように、アクティビティ名やビューの ID は各自変更してください。また、赤文字と赤線がないようにデバッグを行ってください。

9.2.2 コードの解説:

- findViewById(R.id.themeSwitch)

  XML レイアウトファイルで定義された Switch の ID を使って、対応する Switch オブジェクトを取得します。

- themeSwitch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {...}

  スイッチの状態が変更されたときに実行する処理を記述します。

- AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES)

  ダークテーマに切り替えます。

- AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO)

  ライトテーマに切り替えます。

- Toast.makeText(MainActivity.this, ...).show()

  スイッチの状態が変更されたときにテーマの変更を通知する Toast メッセージを表示します。

9.2.3 デバッグのポイント:

- テーマが正しく切り替わらない場合、スイッチの ID が正しいか確認しましょう。
- スイッチの状態変更イベントが正しく設定されているか確認しましょう。

9.3 完成したプログラム

完成したプログラムは以下の通りです。

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main3);

        // レイアウト内のEditTextとButtonの参照を取得
        EditText nameEditText = findViewById(R.id.editTextText2);
        Button greetButton = findViewById(R.id.button2);
        TextView greetingTextView = findViewById(R.id.textView4);

        // ボタンのクリックイベントを設定
        greetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 入力された名前を取得
                String name = nameEditText.getText().toString();

                // メッセージを表示
                greetingTextView.setText(name + "さん、こんにちは！");

                // Toastでメッセージを表示
                Toast.makeText( context: MainActivity3.this,  text: name + "さん、こんにちは！", Toast.LENGTH_SHORT).show();
            }
        });

        // スイッチの参照を取得
        Switch themeSwitch = findViewById(R.id.switch2);
        // スイッチの状態変更イベントを設定
        themeSwitch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                if (isChecked) {
                    // ダークテーマに切り替え
                    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);
                } else {
                    // ライトテーマに切り替え
                    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
                }

                // Toastでテーマの変更を通知
                String theme = isChecked ? "ダークテーマ" : "ライトテーマ";
                Toast.makeText( context: MainActivity3.this,  text: theme + "に変更されました", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

9.4 まとめ

自己紹介アプリに機能を追加することで、実際に動作するアプリを作成することができました。これにより、Java の基本文法だけでなく、Android アプリの開発に必要な基礎的な技術も学ぶことができました。アプリ開発では、コードの正確さやデバッグが非常に重要です。今回のセクションで学んだ内容を活かして、さらに高度なアプリ開発に挑戦していきましょう。次回は、自力で簡単な java のプログラムを書いていきます。