**プログラム解説書**

**概要**

このプログラムは、ユーザーが4桁の数字を入力し、それをランダムに生成された4桁の数字と比較して“○（正解位置と数字一致）”、“×（不一致）”、“△（数字のみ一致）”の結果を表示するゲームアプリケーションです。

**ファイル構成**

1. **App.java**
   * ゲームロジックとGUIの設定が記述されています。
2. **Main.java**
   * アプリケーションを起動するためのエントリポイントです。

**各ファイルの詳細**

**1. App.java**

**主なクラスとフィールド**

* **JFrame**: AppクラスはJFrameを継承し、ウィンドウ全体の管理を行います。
* **JPanel**: ウィンドウの各部分を分割するためのコンポーネント。
* **JTextField**: ユーザー入力を表示します。
* **JLabel**: ゲームの結果や“○、×、△”のカウントを表示します。
* **JButton**: ユーザーが数字や“回答”を入力するためのボタン。

**初期化処理**

* **ランダムな数字の生成**:
* RND1 = (int) (Math.random() \* 10);
* RND2 = (int) (Math.random() \* 10);
* RND3 = (int) (Math.random() \* 10);
* RND4 = (int) (Math.random() \* 10);

これにより0から10までの数字が4つ生成されます。この数字はゲームロジックにおいて、ユーザーが入力する数字と比較される「正解」として使用されます。各数字は一桁のランダムな値で、入力された数字がこれらと一致するかどうかで結果が決まります（例: 正確な位置と数字が一致すれば“○”）。

* **GUIの設定**: 各JPanelの役割と配置場所は以下の通りです：
  + **panel1**: ユーザーの入力（JTextField）と結果表示（JLabel）を担当します。このパネルはメインウィンドウの左上に配置されます。
  + **panel2**: “○、×、△”のカウント表示を担当します。このパネルはメインウィンドウの右上に配置されます。
  + **panel3**: ユーザーが数字を入力するためのボタンを含むパネルです。このパネルはメインウィンドウの左下に配置されます。
  + **panel4**: 回答ボタンを配置するパネルです。このパネルはメインウィンドウの右下に配置されます。
* **GridLayoutについて** GridLayoutは、GUIコンポーネントを均等なサイズのグリッド状に配置するためのレイアウトマネージャです。主な利点として、簡潔に一貫したデザインを保つことができる点が挙げられます。このプログラムでは、以下のように使用されています：
  + **利点**:
    - 均等なセルサイズが自動的に計算されるため、手動調整が不要。
    - レイアウトが簡潔に定義でき、コードの可読性が向上。
  + **他のレイアウトとの比較**:
    - BorderLayoutは、指定された位置にコンポーネントを配置できますが、セルサイズが不均一になる場合があります。
    - FlowLayoutは、コンポーネントを左から順に並べますが、ウィンドウサイズによって配置が変動します。
  + **使用例**:
    - panel1には2行1列のグリッドを設定し、テキストフィールドとラベルを縦に並べています。
    - panel2には2行3列のグリッドを設定し、結果表示用のラベル（○、×、△）とそのカウントを表示しています。
    - panel3には4行3列のグリッドを設定し、数字入力用のボタンを配置しています。

**メインロジック**

* **数字入力処理**: 数字入力処理の流れは以下の通りです：
  1. ボタンが押される。
  2. ボタンのラベル（文字列）を取得する。
  3. 取得したラベルを整数に変換し、answer配列に追加する。
  4. 入力された数字をJTextFieldに追加して表示する。
* if (e.getSource() == key1) {
* String str = key1.getText();
* answer[answerCount] = Integer.parseInt(str);
* answerCount++;
* textField1.setText(textField1.getText() + str);
* }
* if (e.getSource() == key1) {
* String str = key1.getText();
* answer[answerCount] = Integer.parseInt(str);
* answerCount++;
* textField1.setText(textField1.getText() + str);
* }
* **addActionListenerメソッドについて**: addActionListenerメソッドは、ボタンが押されたときに実行される処理（イベントリスナー）を登録するために使用されます。このプログラムでは、すべてのボタンに対してこのメソッドを使用しており、actionPerformedメソッドが呼び出される仕組みになっています。

**どのコンポーネントによって呼び出されたかを判定する処理**: ActionEventオブジェクトのgetSourceメソッドを使用して、押されたボタンを特定します。たとえば：

if (e.getSource() == key1) {

// key1ボタンが押された場合の処理

}

この判定により、どのボタンが押されたかを確認し、対応する処理を実行できます。

* **actionPerformedメソッドのオーバーライドが必要な理由**: actionPerformedメソッドは、ActionListenerインターフェースで定義されている抽象メソッドです。このため、ActionListenerを実装するクラスでは、このメソッドを必ずオーバーライドしなければなりません。オーバーライドしないと、コンパイルエラーが発生します。オーバーライドすることで、特定のイベント（この場合はボタンのクリック）に対する処理を自由に定義できます。
* **回答ボタン処理**: 4桁の入力が完了した後、ランダムな数字と比較し、“○、×、△”を計算します。
* if (RND1 == answer[0]) {
* maruCount++;
* } else if (RND1 == answer[1] || RND1 == answer[2] || RND1 == answer[3]) {
* sankakuCount++;
* } else {
* batuCount++;
* }

**結果表示とリセット処理**

* **結果表示**: 比較結果がJLabelに反映され、ユーザーにフィードバックされます。
* maru2.setText(String.valueOf(maruCount));
* batu2.setText(String.valueOf(batuCount));
* sankaku2.setText(String.valueOf(sankakuCount));
* **リセット処理**: ゲームの状態をリセットするために以下の処理を行います：
* maruCount = 0;
* batuCount = 0;
* sankakuCount = 0;
* textField1.setText("");
* answerCount = 0;
  + maruCount, batuCount, sankakuCount を0にリセットして、新しいゲームに備えます。
  + textField1 を空文字列に設定して、ユーザーの入力フィールドをクリアします。
  + answerCount を0に戻し、新しい入力を受け付ける準備をします。

**2. Main.java**

* **エントリポイント**: Appクラスのインスタンスを生成し、アプリケーションを起動します。
* public static void main(String[] args) {
* App app = new App();
* }

**実行手順**

1. プログラムをコンパイルします。
2. javac App.java Main.java
3. 実行します。
4. java Main