7-12.例外処理

例外処理の基本構文

例外処理の記述は、`try-catch-finaly`文によって行いました。

```
try {
    // 例外発生の可能性がある処理
} catch {
    // 例外が発生した時の処理
} catch {
    // 例外が発生した時の処理
} finally {
    // 例外発生の有無に関わらず実行したい処理
}
```

`try`部で例外が発生すると、すぐに`catch`ブロックに制御が移ります。
しかし、処理を途中で中断してしまうと、ネットワーク接続の切断処理やファイルのクローズなど、必ず行わなければいけない処理を行えない可能性があります。
そのような、例外発生の有無に関わらず必ず実行したい処理を、`finally`ブロックに記述します。

また、`try-catch-finally`構文を、`catch-try-finally`など別の順番に変更することは出来ません。
それを行うと、コンパイルエラーが発生します。

また、`try-catch-finally`構文の中で、複数回記述できるのは`catch`だけです。
`try`や`finally`を2回以上記述すると、コンパイルエラーとなります。

上記は問題で問われることがあるので注意しましょう。

到達できないcatch部

以下のコードを見てみましょう。

```
public class Main {
    public static void main(String[] args) {
        try {
            int[] numbers = { 1, 2, 3 };
            System.out.println(numbers[5]); //例外発生

        } catch (Exception e) {
            System.out.println("例外です");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("配列の範囲外にアクセスしています");
        }
    }
}
```

配列変数`numbers`に3つの値が入っており、println で要素にアクセスしようとしますが、 **インデックス5は配列内に存在せず、例外が発生します。**

このような、配列の範囲外にアクセスした際は、`ArrayIndexOutOfBoundsException`の例外が発生し、catch部で受け取ろうとしています。

しかし、その例外の前のcatch部として、`Exception`を受け取るコードが書かれています。
`Exception`クラスは全ての例外のスーパークラスであり、つまり全ての例外を受け取るクラスになります。

発生した例外は、全てこの場所で受け取られます。
つまり、`ArrayIndexOutOfBoundsException`のcatch部にはどうしても到達できません。
このような構造の場合、コンパイルエラーが発生しますので覚えておきましょう。

returnの挙動

次のコードの出力結果を考えてみて下さい。

?

```
public class Main {
    public static void main(String[] args) {
        String test = sample();
        System.out.println(test);
    }

    private static String sample() {
        try {
            int[] numbers = { 1, 2, 3 };
            System.out.println(numbers[5]); //例外発生
        } catch (ArrayIndexOutOfBoundsException e) {
            return "A";
        } finally {
            System.out.println("B");
        }
        return "C";
    }
}
```

`try`部で例外が発生し、`catch`部で受け取りますが、 **その処理として`return`が書いてあります。**
そしてその後、`finally`部に出力処理があります。

`return`は、メソッドの実行場所であるmainメソッドに戻り値を返します。
そのように、前半のコードに処理を抜ける記述があったとしても、必ず`finally`部は実行されるので注意しましょう。

今回の場合、 **`return`で`A`を返そうとし、そこで`finally`の処理が動き、戻り値が返る** という順序になりますので、出力は以下のようになります。

```
//出力結果
B
A
```

また、次のコードを考えてみます。

```
public class Main {
    public static void main(String[] args) {
        String test = sample();
        System.out.println(test);
    }

    private static String sample() {
        try {
            int[] numbers = { 1, 2, 3 };
            System.out.println(numbers[5]); // 例外発生
        } catch (ArrayIndexOutOfBoundsException e) {
            return "A";
        } finally {
            return "B";
        }
    }
}
```

`catch`と`finally`のどちらにも`return`がありますが、どちらが優先されるでしょうか。

このように、2回の戻り値が返された時は、後の値で上書きするような処理が行われます。
(変数の上書きのようなイメージです。)

上記コードでは、まず`catch`部で例外を受け取り、`return "A";`を実行します。
その後、必ず処理を行う`finally`部の`return "B";`を実行します。
よって、mainメソッドには`B`が返され、出力されます。

```
//出力結果
B
```

以上のように、例外処理の記述が正しくできているか、正しい記述はどれかを問われる問題が出てきます。
知らなかった仕様があれば自分でも試して確かめてみましょう！

課題

満点を取れるまで小テストを受験して下さい。

制限時間: 30 分

評定方法: 最高評点

合格点: 100 / 100

受験件数: 60