7-6.条件式

if文　波カッコの省略

if文の基本構造は以下のようなものでした。

```
if ( 条件式 ) {
    一致した時に行う処理
}
```

この構造の中で、波カッコ `{ }`は省略して書くことが出来ます。

```
public class Main {
    public static void main(String[] args) {
        int num = 10;
        if (num <= 10)
            System.out.println("JAVA");
    }
}

//出力結果
JAVA
```

通常のコーディングでは、波カッコの省略は推奨されません。
構造が分かりづらくなり、人が読んだ時の可読性が下がるためです。
ただし、試験ではその仕様を問う問題が出題されます。

次のコードの出力結果はどうなるでしょうか。
結果として正しい選択肢を1つ選択して下さい。

```
public class Main {
    public static void main(String[] args) {
       if (false)
       System.out.println("A");
       System.out.println("B");
    }
}
```

A. 「A」が表示される
B. 「B」が表示される
C. 「A」「B」が表示される
D. 何も表示されない

先程と同様に、ifの波カッコが省略されています。
ただし、どこまでがifの処理内容に含まれているかには注意が必要です。
波カッコが省略されていた場合、次の1文のみがその処理内容として判断されます。

上記を書き直すと以下のようになります。

```
public class Main {
    public static void main(String[] args) {
        if (false) {
            System.out.println("A");
        }
        System.out.println("B");
    }
}
```

Aの出力文はifの中、Bの出力文はifの外になります。
今回のifの条件式として初めから`false`が記述されてますので、この中身は実行されません。
よって、ifの枠外にある`System.out.println("B");`だけが実行されるため、
出力結果は`B. 「B」が表示される`となります。

ifだけでなく、`else if`や`else`の波カッコも省略することができます。

?

```
public class Main {
    public static void main(String[] args) {
        int num = 10;
        if (num == 100)
            System.out.println("A");
        else if (num < 10)
            System.out.println("B");
        else if (num == 10)
            System.out.println("C");
        else if (num == 10)
            System.out.println("D");
    }
}
```

これにカッコを付けると以下の通りとなります。

```
public class Main {
    public static void main(String[] args) {
        int num = 10;
        if (num == 100) {
            System.out.println("A");
        } else if (num < 10) {
            System.out.println("B");
        } else if (num == 10) {
            System.out.println("C");
        } else if (num == 10) {
            System.out.println("D");
        }
    }
}

//出力結果
C
```

しかし、次はどうでしょうか？

```
public class Main {
    public static void main(String[] args) {
        int num = 10;
        if (num == 100)
            System.out.println("A");
        else if (num < 10)
            System.out.println("B");
        else
        if (num == 10)
            System.out.println("C");
        else
        if (num == 10)
            System.out.println("D");
    }
}
```

途中の`else if`が改行されています。
改行してしまうと、`else if`とは扱われないので注意が必要です。
elseがあり、その処理内容のコードとしてifがある、と解釈されます。
上記を書き直すと、以下のようになります。

```
public class Main {
    public static void main(String[] args) {
        int num = 10;
        if (num == 100) {
            System.out.println("A");
        } else if (num < 10) {
            System.out.println("B");
        } else {
            if (num == 10) {
                System.out.println("C");
            } else {
                if (num == 10)
                    System.out.println("D");
            }
        }
    }
}

//出力結果
C
```

出力は変わらずCとなりますが、先程とコードの構造が違うので見比べてみて下さい。

switch文のフォールスルー

switch文の基本構造は、下記のようなものでした。

```
public class Main {
    public static void main(String[] args) {
        int num = 2;
        switch(num) {
            case 1:
                System.out.println("A");
                break;
            case 2:
                System.out.println("B");
                break;
            case 3:
                System.out.println("C");
                break;
            default:
                System.out.println("D");
                break;
        }
    }
}

//出力結果
B
```

この中で処理を中断させるbreakは、記述は必須ではありません。
break無しで書く書き方を、 **フォールスルー** と呼びます。

このフォールスルーの書き方の仕様は注意が必要です。

```
public class Main {
    public static void main(String[] args) {
        int num = 2;
        switch(num) {
            case 1:
                System.out.println("A");
            case 2:
                System.out.println("B");
            case 3:
                System.out.println("C");
            default:
                System.out.println("D");
        }
    }
}

//出力結果
B
C
D
```

変数numの値は2なので、case 2で一致します。
なのでcase 2の時の処理としてBを出力しますが、 **その後breakが見つからない限り、全ての記述を処理していきます。**

すぐ下の条件、case 3には当てはまりませんが、それとは関係なく、Cを出力する処理がなされます。
どのcaseにも当てはまらない場合のdefaultも同様に処理の対象となるので気をつけましょう。

次はどうでしょうか。

```
public class Main {
    public static void main(String[] args) {
        int num = 5;
        switch(num) {
            default:
                System.out.println("A");
            case 1:
                System.out.println("B");
            case 2:
                System.out.println("C");
                break;
            case 3:
                System.out.println("D");
        }
    }
}
```

numの値は5で、どのケースにも当てはまりません。
なのでdefaultの場所が始めに参照されます。
default**は一番最後に書くことが多いですが、どの場所にあっても問題有りません。**

defaultの場所から、Aの出力、Bの出力、Cの出力を行います。
Cの出力をした後に、breakが見つかります。なのでここでストップします。

```
//出力結果
A
B
C
```

switch文は、カッコの中の式とcaseの値を比較しますが、 **単に処理に入るための入口を探しているようなものだと捉えておきましょう**
入り口が見つかったら、そこから1行ずつ処理を進める。breakがあったらストップする。
この認識を持っていれば対応できるかと思います。

switchのcaseに設定できる値

次のswitch文に設定されたcase部のうち、 **コンパイルエラーが起きる場所が2箇所あります。**

```
public class Main {
    public static void main(String[] args) {
        final int NUM = 0;
        int num = 10;
        switch (num) {
            case "10":
                System.out.println("A");
                break;
            case num:
                System.out.println("B");
                break;
            case 2 * 5:
                System.out.println("C");
                break;
            case NUM:
                System.out.println("D");
                break;
        }
    }
}
```

switch文のcaseに指定できる値の条件として、以下を満たす必要があります。

- 条件式が戻す値と同じ型か互換性がある型であること
- nullでないこと
- 定数であるか、コンパイル時に値を決めることができること

よって、上記のコードでエラーが起きる場所は、`case "10":`と`case num:`になります。

条件式(カッコの中身)と同じ型かどうか、またnullでないかどうかは覚えやすいかと思います。
型が合っていないと、そもそも一致するはずがないので記述できないルールになっています。

試験対策で気をつけておくべきは、3つ目の **「定数であること」** です。
final宣言された変数 か、 **普通に記述された値(リテラル)** であることを指します。
これは、最初にcaseを記述した時から変更できる状態だと、2つのcase値が同じになってしまうなどの問題が起きる可能性があるためこのような仕様になっています。
普通の変数はcaseに設定できませんので覚えておきましょう。

switch文　条件式に設定できるデータ型

switch文は、`switch(条件式)`の記述から始まりますが、ここの条件式に設定できるデータ型の種類には制限があります。
次のうちの型でなければエラーが発生します。

- char
- Character (charのラッパークラス)
- byte
- Byte (byteのラッパークラス)
- short
- Short (shortのラッパークラス)
- int
- Integer (intのラッパークラス)
- String
- Enum (列挙型)

種類が多いですが、

- int型以下の整数型とそのラッパークラス
- 文字と文字列
- 列挙型
  とおぼえておきましょう。

ラッパークラスとは
enumとは

間違いやすいのは`long`型が入っていないこと、`double`や`float`などの小数点を扱う型が入っていないことです。
`boolean`型も含まれていません。

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: 88