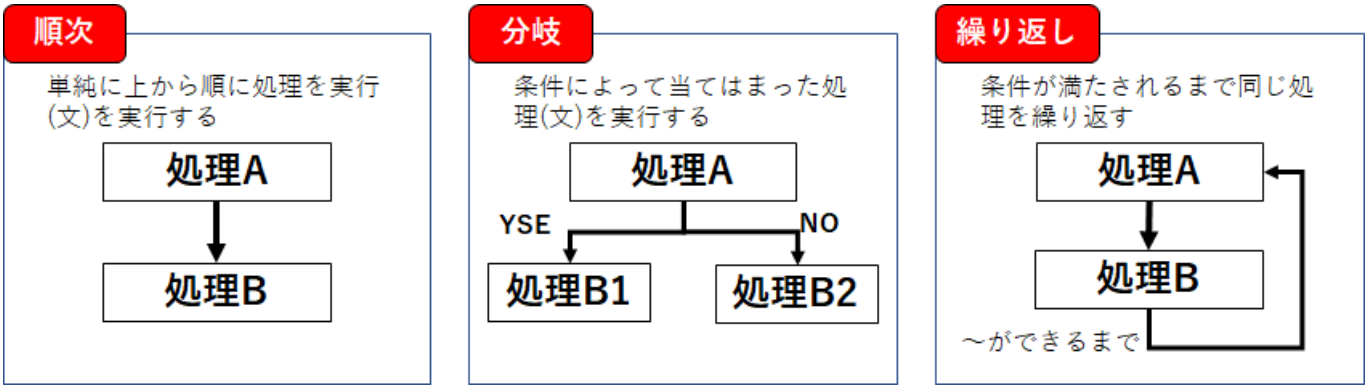
プログラムの流れ

代表的な制御構造

基本的にプロクラムが読まれる処理の順番が「 **上から順に1つずつ** 」実行されるのがルールとなっています。文を実行させる順番のことを **制御構造** (制御フロー)と呼び、代表的なものが以下の3つとなります。

1. 順次
2. 分岐
3. 繰り返し



1. 順次

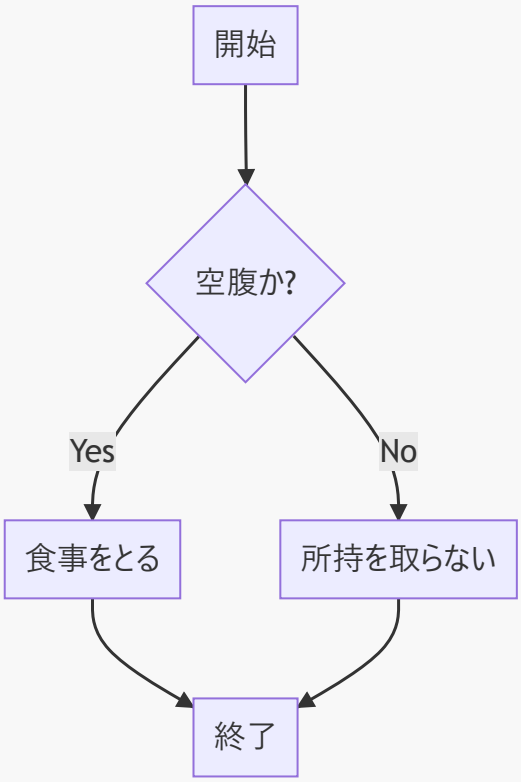
基本的な処理となる制御構造です。上記の画像上の処理Aに「分岐」や「繰り返し」処理が入るかもしれません。
この制御構造の上に様々の制御(プログラミング)が記述され処理されていきます。

2. 分岐

条件によって次に行いたい処理を分岐させる制御構造となります。
日常生活で例えるなら、

- もし「お腹が空いたら」食事をとる
- もし「お腹が空いていないなら」食事をとらない

このように**もし**「お腹が空いていたら」「お腹が空いていなかったら」でそのあとの「食事をとる」「食事をとらない」といった様に次に行う処理を **条件によって変化させている** 制御構造になっています。



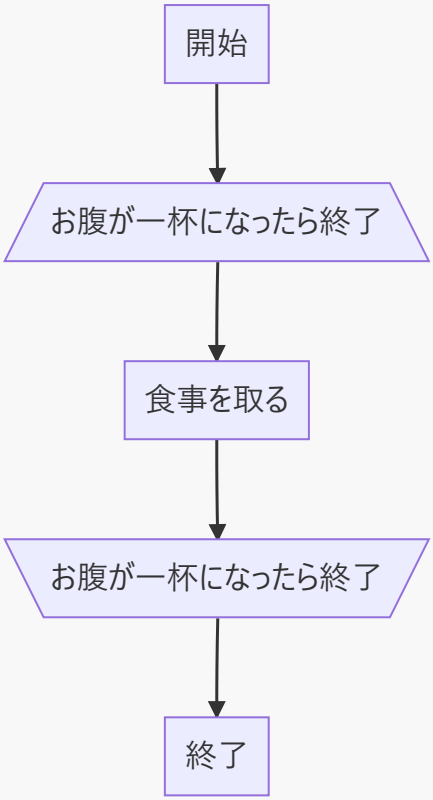
3. 繰り返し

条件が満たされるまで同じ処理をさせる制御構造となります。

日常生活で例えると、「お腹が空いている」状態が条件だとします。すると

- 食事を取る

この処理が条件を満たすまで何度も同じ処理を **繰り返し行い条件が満たされると処理が終了する** 制御構造となっています。



分岐処理(if文)を書いてみる

2. 分岐でも記載しましたが分岐処理は「**もしも～の場合**」は「この処理」をしますという動作になります。
基本的な構文で記述すると以下の様になる。

```
if (条件) {
    処理
}
```

- `if`
  分岐を指定する命令後
- (条件)
  分岐の条件を記述する。**条件として、booleanのture/falseの状態になる条件しか記述することができません。**

こんな例題があったとします。

> 国語、数学、英語の試験の点数の合計によって「合格/補習/不合格」を判定して表示するプログラムを作成してください。
> 以下が条件になります。
> ・合格: 合計点が240点以上の場合
> ・補習: 合計点が240点未満で150点以上の場合
> ・不合格: 150点未満の場合

この条件を満たすプログラミングを以下の様に作成してみました。
if文で条件式を3記述することで例題の条件を満たすことができます。

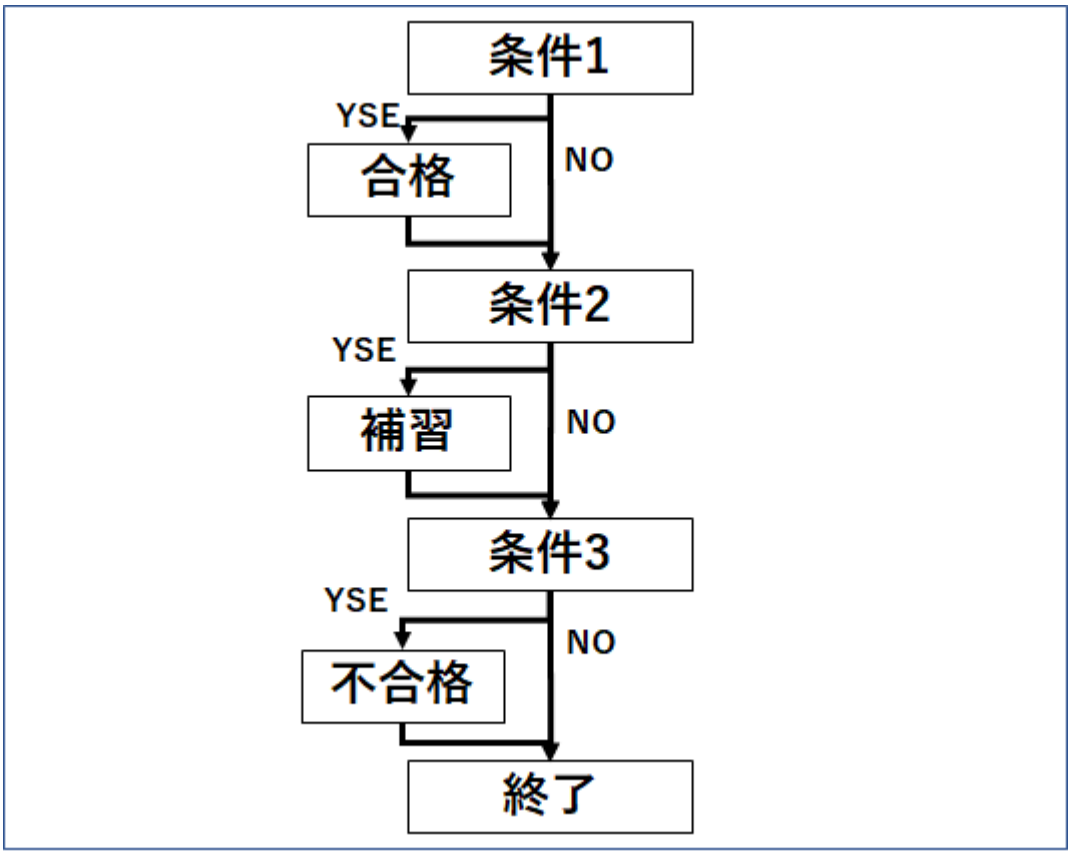
if文のみを使用して記述した場合

```
public class Main {
    public static void main(String[] args) {
        // 試験の点数によって合格/補習/不合格かを判定するコード
        // 国語の点数
        int kokugo = 50;
        // 数学の点数
        int sugaku = 50;
        // 英語の点数
        int eigo = 50;
        // 合計点: 150
        int goukei = kokugo + sugaku + eigo;

        // 条件式
        // 240点以上なら合格
        // 150点~240点未満なら補習
        // 150点未満なら封合格
        if(goukei >= 240) { // 条件式1
            System.out.println("合格");
        }
        if(goukei >= 150 && goukei < 240) { // 条件式2
            System.out.println("補習");
        }
        if(goukei < 150){ // 条件式3
            System.out.println("不合格");
        }

    }
}
```

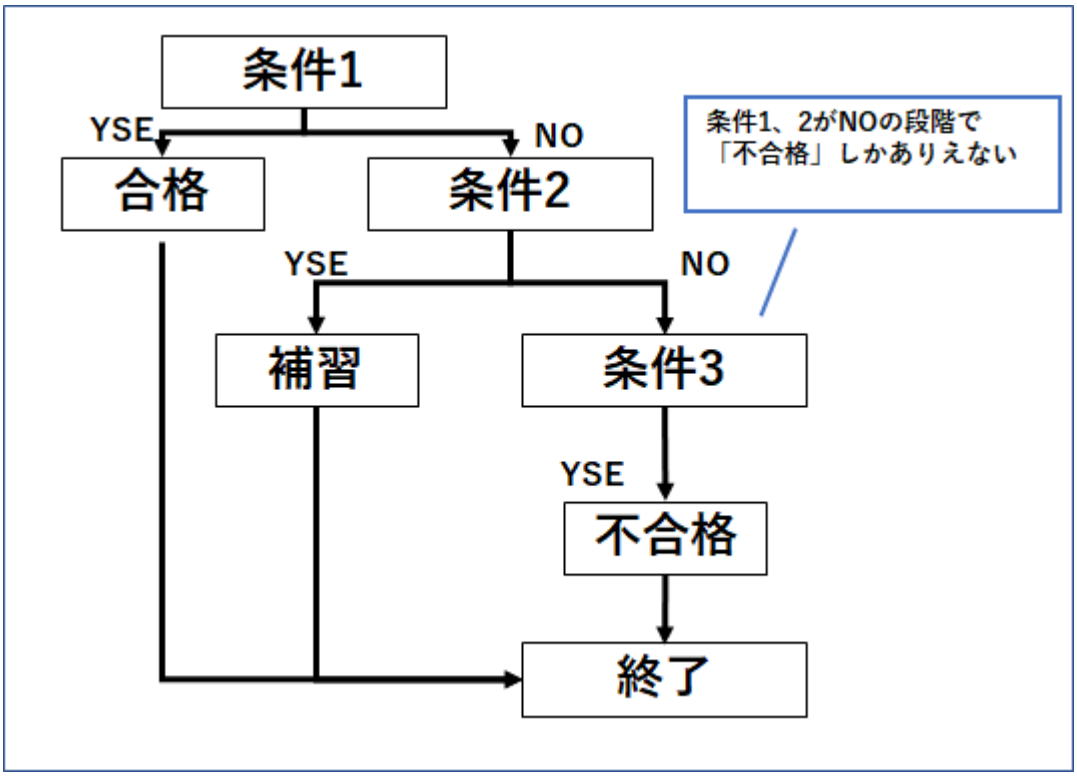
今回の処理手順を図にしてみると以下の様になります。



処理の順序は制御工場の順次により条件式1, 2, 3の順番で判定が行われます。今回は、`goukei`は150点のため条件式2のif文が実行されることになります。
しかし、仮に条件式2の'if(goukei >= 150 && goukei < 240)'の以上未満の条件を間違えて'if(goukei <= 150 && goukei < 240)'**240点未満と150点未満の条件を満たした場合**になってしまいますと条件式2と条件式3が実行されてしまいます。

else文なども併用して記述した場合

if文にも条件式をまとめることができます。今回の例題の場合、合計点数という同じ条件の一定点数の以上未満を判定しています。そして、例題の性質上「合格/補習/不合格」が2つ表示されることはありえない状態です。
処理手順をもっと単純に考えてみましょう。



このように条件式1が満たされた場合、下の条件式を判定する必要がなくなります。
この様な場合のif文の記述すると以下の様になります。

```
if(条件式1){
    //処理1
} else if(条件式2){
    // 処理2
} else {
    // 処理3
}
```

- `if(条件)`
  条件を満たすと、下記に記載している`else if`、`else`の条件式はスキップされます。
- `else if(条件)`
  `if(条件)`の判定が`false`(条件を満たせなかった)の場合、上から順に`else if`を判定していき条件を満たすと、それ以下の条件式はスキップします。
- `else`
  上記の条件がすべて満たせなかった場合、必ず実行されます。

```
public class Main {
    public static void main(String[] args) {
        // 試験の点数によって合格/補習/不合格かを判定するコード
        // 国語の点数
        int kokugo = 80;
        // 数学の点数
        int sugaku = 50;
        // 英語の点数
        int eigo = 60;
```

```
        // 合計点
        int goukei = kokugo + sugaku + eigo;

        // 条件式
        // 240点以上なら合格
        // 150点~240点未満なら補習
        // 150点未満なら封合格
        if(goukei >= 240) { // 条件式1
            System.out.println("合格");
        } else if(goukei >= 150) { // 条件式2
            System.out.println("補習");
        } else { // 条件式3
            System.out.println("不合格");
        }

    }
}
```

上記のコードの条件式を見るとgoukeiは190点となってます。
まず、制御構造の順次により条件式1, 2, 3の順番で判定が行われます。
条件式1ではgoukeiが190 >= 240(240点以上)を満たしていない、つまりfalseの判定になるため、{}の処理はスキップされます。
次に条件式2の190 >= 150(150点以上)は満たしている、つまりtrueの判定になるため、{}の処理が実行されます。
そして、条件式2が成立したことによりelseの条件式3はスキップされます。

繰り返し処理を書いてみる

繰り返し(ループ)処理の基本的な構文は以下の3種類となります。

- while
- do-while
- for

while文

基本的な記述方法は以下の様になります。

```
while(条件式){
    処理
}
```

if文の時と同じ様に条件式を満たす場合(true)は{}ブロックに記述されている処理を実行し続けます。なので、条件式の設定を失敗すると **無限ループ** (処理が永久に終わらない)になってしまう可能性があるので注意しましょう。

> JDKを利用している場合は、「Ctrl」+「c」で強制終了させることができます。

こんな例題があったとします。

> 水槽を満たすための水は100L必要です。
> Aさんが1回に運べる量は20Lとします。
> この動作をプログラムで記述してください。
> なお、「Aさんは??回目で合計??L運びました。」を繰り返す度に表示させてください。

```
public class Main {
    public static void main(String[] args) {
        // 水槽に入る水の最大量
        int max = 100;
        // 運べる水の量
        int carry = 20;
        // 運んだ水の量
        int total = 0;
        // 運んだ回数
        int count = 0;

        // 運んだ水の量が100L未満の状態の時はブロック({処理})内の処理を行う
        while (max > total) {
            count++;
            total += carry;
            System.out.println("Aさんは" + count + "回目で合計" + total + "L運びました。");
        }
    }
}
```

do-while文

基本的な考え方は、while文と同様ですが記述方法が異なります。

```
do {
    処理
} while(条件式)
```

while文と同じ例題を記述すると、

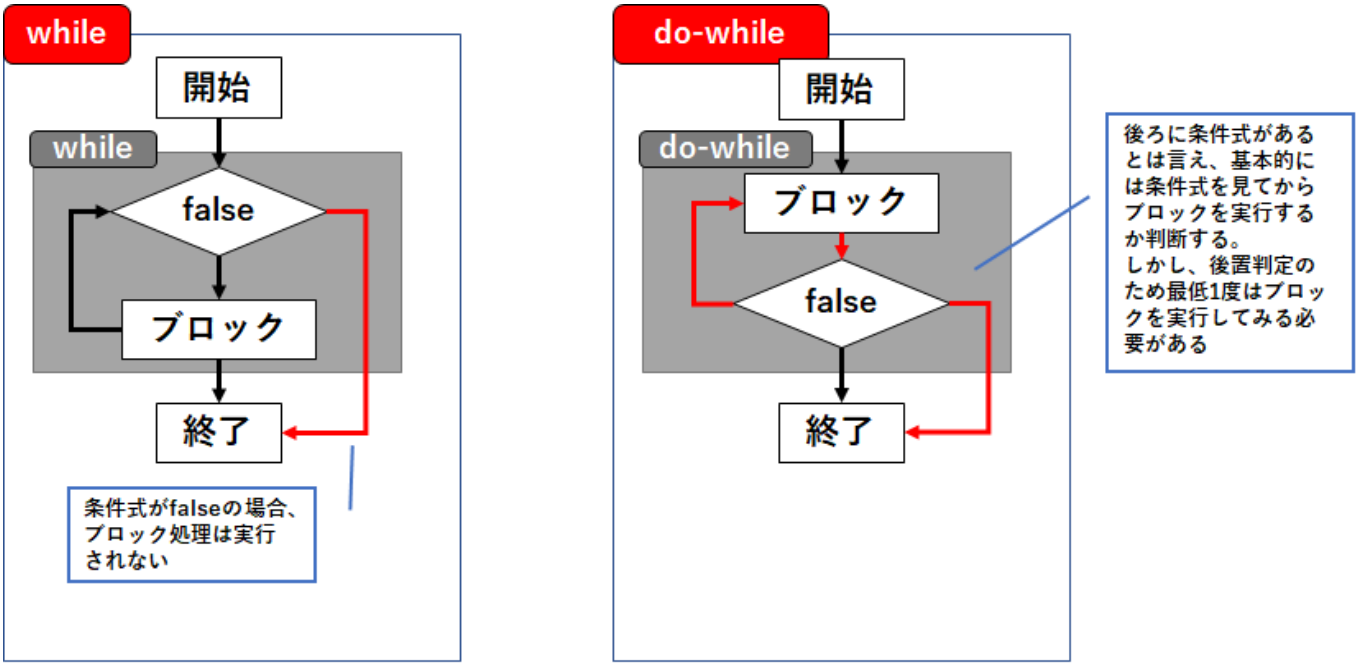
```
public class Main {
    public static void main(String[] args) {
        // 水槽に入る水の最大量
        int max = 100;
        // 運べる水の量
        int carry = 20;
        // 運んだ水の量
        int total = 0;
        // 運んだ回数
        int count = 0;

        // 運んだ水の量が100L未満の状態の時はブロック({処理})内の処理を行う
```

```
            do {
                count++;
                total += carry;
                System.out.println("Aさんは" + count + "回目で合計" + total + "L運びました。");
            } while (max > total);
        }
    }
```

while文と全く同じ処理ができます。

しかし、見ての通り条件判定を行う場所が後ろになっています。これを **後置判定** と呼びます。これは、**最低1回はブロック内の処理を実行する** ことを意味します。



条件式で判断してブロックを処理するか決定する

for文

今までのループ文は、「ある条件が成立するまで繰り返す」ことを主な目的に使用しますが、 今回の **for文** は「～回繰り返す」ことを主な目的として使用する記述方法になります。

基本的な記述方法は以下の様になります。

```
for(ループ変数; 繰り返し条件; 繰り返し時処理) {
    処理
}
```

- ループ変数
  for文が開始されいる時に最初に1回だけ実行される文です。 **通常、「何週目のループかを記録しておお**

く変数」 を定義します。

- 繰り返し条件
  ブロックの内容を実行する前に評価され、ループを継続するかどうかを判定する条件式です。評価結果が`true`の間はブロック内の処理が繰り返し実行されます。
- 繰り返し時処理
  for文内のブロックを最後まで処理して、`}`まで **到達した直後に自動的に実行される文** となります。通常は、「`i++`」のようにループ変数値を1だけ増やす文を書きます。

こんな例題があったとします。

> 1から10までの合計値を表示するプログラムを作成してください。
> 計算過程の表示は必要ありません。

```
public class Main {
    public static void main(String[] args) {
        // 計算結果を入れる変数
       int total = 0
       /*
       ループ変数iが1で初期化される
       ブロック処理が終了時にiに1加算される
       iの10になったらループ処理が終了する
       */
       for(int i = 1; i <= 10; i++){
           total += i;
       }

       System.out.println("1から10までの合計値は" + total + "です");
}
```

制御構文の応用

制御構造のネスト

順次でも記載した通りすべての処理順番の基本系となります。
また「分岐」や「繰り返し」の制御構造は、その中の制御構造を含むことができ、**「分岐の中に分岐」** や **「繰り返しの中の分岐」** の様な実装することができ、このような多重構造を **入れ子** や **ネスト** と呼びます。

例えば、for文による繰り返しをネストさせて、「九九の表」を出力してみましょう。

```
public class Main {
    public static void main(String[] args) {
        // iは1から9を繰り返す
        for(int i = 1; i < 10; i++) {
            // jも1から9を繰り返す
            for(int j = 1; j < 10; j++) {
                // 計算結果を表示
                System.out.print(i * j);
                // 空白を出力
                System.out.print(" ");
```

```
            }
            // ○○の段が終了した時用の改行
            System.out.println("");
        }
}
```

繰り返し処理の中断

繰り返しの途中で、場合によってはその処理を中断されたいことがあります。
その時のために、Javaでは **break文** と **continue文** の2種類の中断方法が用意されています。

- `break`文
  繰り返し自体を中断する構文
  breakを囲んでいる最も内側の繰り返しブロックが即時に中断されます。
- `continue`文
  今実行されている繰り返し処理を中断する構文
  同じ繰り返しの次の周回に進む。

break文の例題

> 次々と人を建物に入場させます。(無限ループ)
> ただ、建物の定員は100人なので定員に達したら処理を止めてください。
> その際に「100人に達したのでこれ以上は定員オーバーになります。」を表示させてください。

```
public class Main {
    public static void main(String[] args) {
        int count = 0;
        // 無限ループの記述方法
        while (true) {
            // 合計が100になったら
            if(count == 100) {
                System.out.println(count + "人に達したのでこれ以上は定員オーバーになりま
す。");
                // 一番内側のブロック、つまり無限ループ自体が中断される
                break;
            }
            // 定員数を1ずつ増加させている
            count++;
        }
}
```

continue文の例題

> 50人の人が建物に入場します。
> 建物には、A部屋とB部屋があり、A部屋に入った人数を数えて表示させます。 また、5の倍数で入場した人はB部屋に入ってもらうので数を数える必要はありません。

```java
public class Main {
    public static void main(String[] args) {
        // 人数を数える
        int count = 0;
        // 50人入場するループ処理
        for(int i = 1;i <= 50;i++) {
            // 5の倍数 = 5で割ってあまりの出ない場合
            if(i % 5 == 0) {
                // count++の処理を行わない
                continue;
            }
            // 入場数を1人ずつ加算する
            count++;
        }
        System.out.println("部屋Aに入ったのは" + count + "人です。");
}
```