



Java基礎研修参考書

2020 年 1 月 9 日

竹下翔悟

株式会社 Wytel

〒153-0043

東京都目黒区東山1-4-13 ASA東山ビル3B

はじめに	3
コンピュータープログラムとは？	3
課題準備	4
・ パスの設定	4
・ Java実行手順	11
Java基礎課題	14
・ 問 1	14
・ 問 2	15
・ 問 3	16
・ 問 4	17
・ 問 5	19
・ 問 6	20
・ 問 7	21
・ 問 8	22
・ 問 9	23
・ 問 1 0	23
・ 問 1 1	26
・ 問 1 2	27
・ 問 1 3	28
・ 問 1 4	28
・ 問 1 5	30
Java基礎知識	31
・ Javaとは	31
・ コンパイル	31
・ 変数	31
・ 型	32
・ 型変換	32
・ if 文	33
・ 演算子とは	34
・ 算術演算子	34
・ 比較演算子	35
・ 論理演算子	35
・ 文字列の比較	37
・ for文	37

• try-catch文	39
• Switch文	40
• 配列	41
• ソートアルゴリズム	43
• バブルソート	43
• 選択ソート	44
• メソッド	45
• アクセス修飾子	46
• 引数と戻り値	47
• オブジェクト指向	48
• インスタンス	49
• コンストラクタ	51
• オーバーロード	52
• 継承	53
• オーバーライド	54
• インターフェース	55
• while文	57
• ループからの抜け方	58
• NULLと空文字	58
• ArrayListクラス	60
• Static修飾子	61
• 修飾子一覧	62

はじめに

本書では、研修生がJava基礎研修課題を進めるうえで参考となる資料を目指して作成しています。疑問点などあれば気軽に講師の方や先輩に聞いてください。

また、間違えている箇所がありましたら気軽に私まで教えてくださると助かります。

皆様の意見でこの資料をより良いものにしていければと思います。

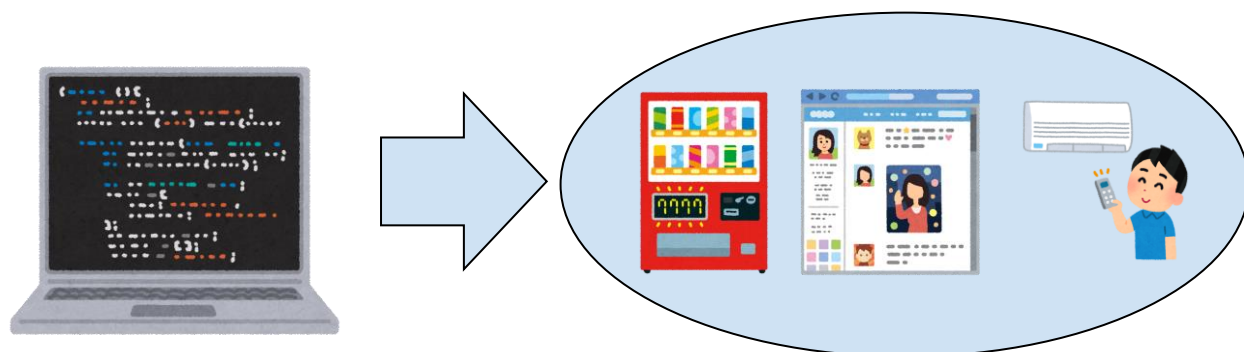


コンピュータープログラムとは？

コンピュータプログラム（以下プログラム）とは、コンピュータに対する命令（処理）を記述したものである。

例えば、自動販売機・ATM・リモコン・スマートフォン等、世の中の多くの物がプログラムを使用して作成されており、基本的には過去に行っていたことをコンピュータによって自動的に行えるようにした物が多いです。

自動化して効率化をする他にインターネットのWebサイトもコンピュータプログラムを使用して作成されています。



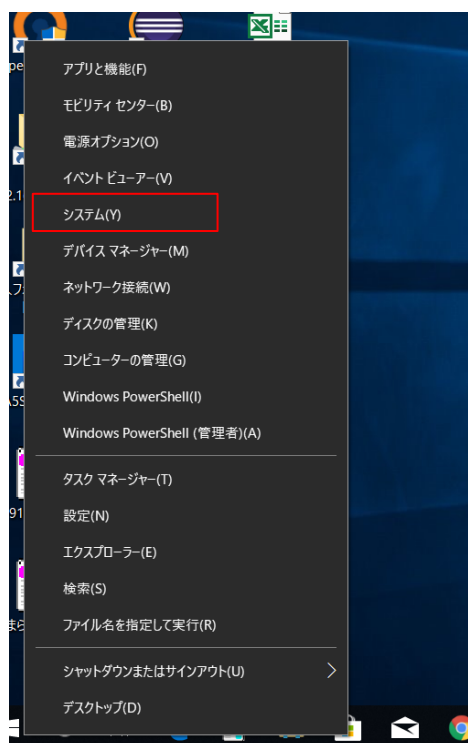
課題準備

・パスの設定

パスの設定とは、インストールしたJDKのコンパイルや実行プログラムを簡単に使えるようにするための設定です。※本書ではWindows10端末を対象に手順を記載してます。

参考サイト：<https://techfun.cc/java/windows-jdk-pathset.html>

①スタートメニューを右クリックし「システム」をクリックします。



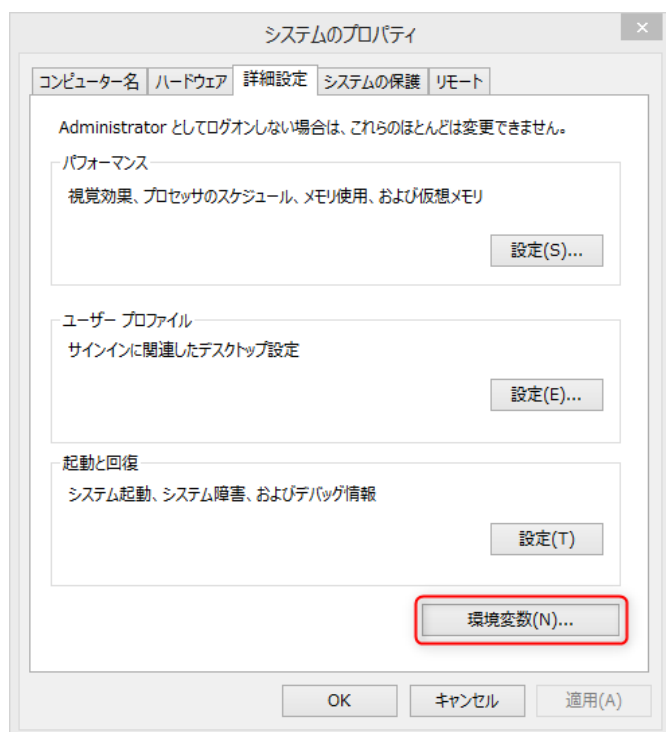
②バージョン情報ウィンドウの「システム情報」をクリック



③「システムの詳細設定」をクリック

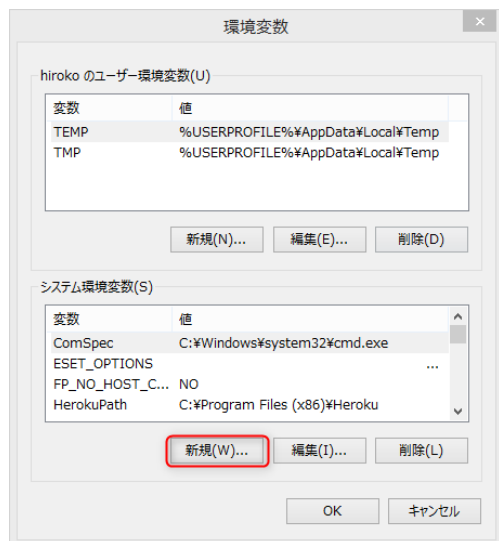


④「システムのプロパティ」画面の「詳細設定」タブを選択し、「環境変数」ボタンをクリックしてください。

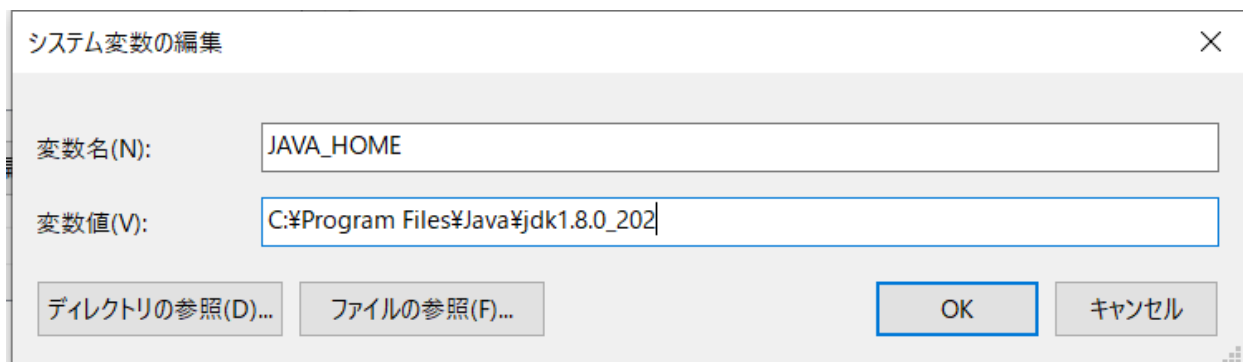


以下の画面でパスの設定を行います。

「新規」ボタンをクリックします。



以下のポップアップ画面に、変数名と変数値を入力してください

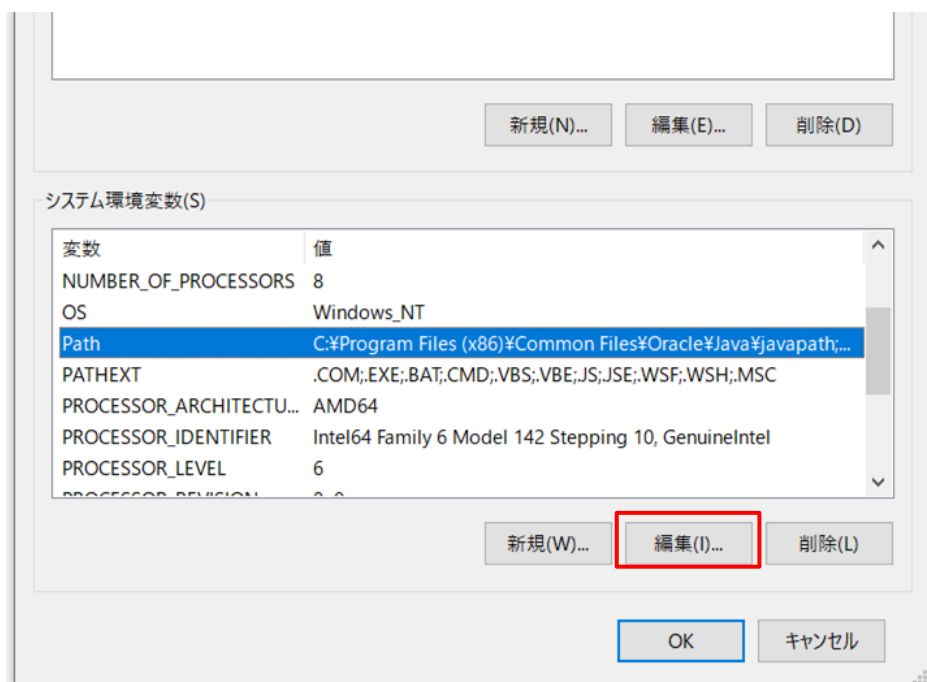


(変数名) JAVA_HOME

(変数値) C:\Program Files\Java\インストールしたjdk

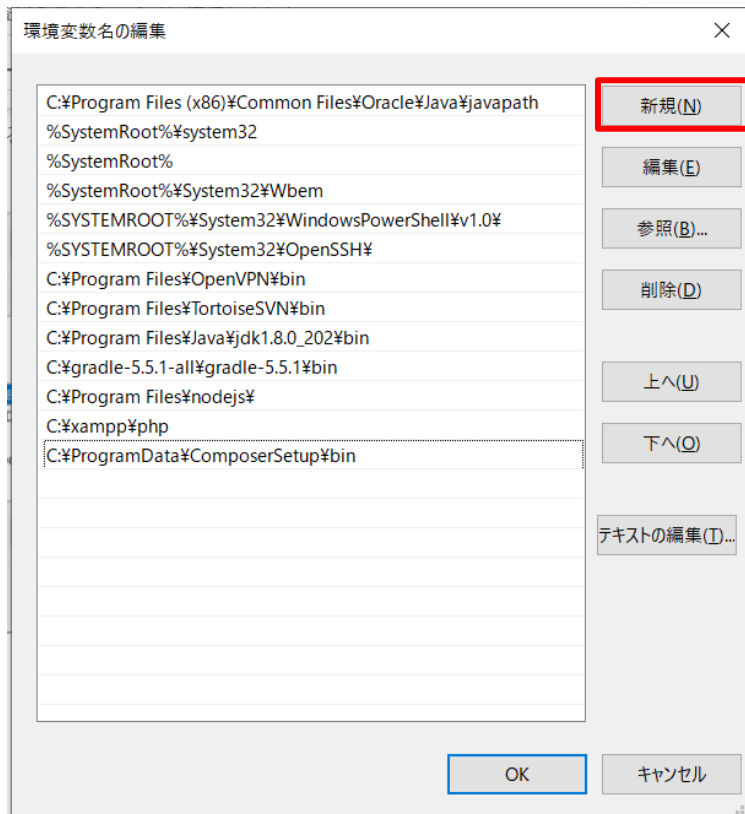
「C:\Program Files\Java」フォルダ内を確認して、最新のバージョン（バージョン番号が大きいもの）を設定してください。

続いてPathを編集します。既存にあるPathを選択し「編集」ボタンをクリックしてください。



新規ボタンを押して以下を入力してください

%JAVA_HOME%\bin;



コマンドプロンプトで以下コマンドを実行してバージョンが正常に表示されることを確認してください。

コマンド : java -version

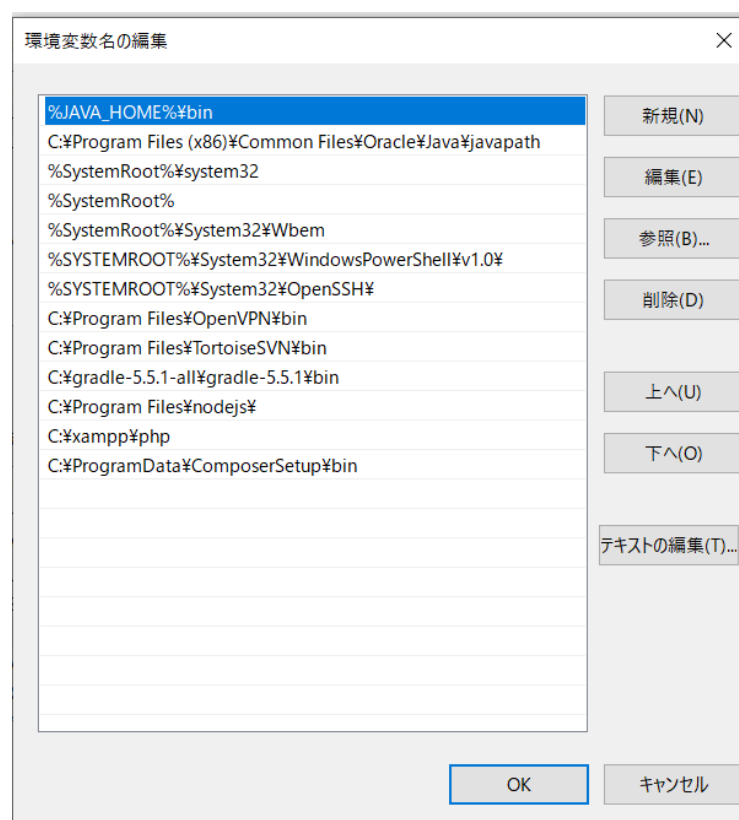
選択C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\takeshita>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)

C:\Users\takeshita>
```

※バージョンが違の場合、[%JAVA_HOME%\bin]を選択し、上へボタンを押して優先順位を上げてみてください。



「javac」 コマンドを入力してオプション一覧が表示されていれば問題はないと思います。

```
C:\Windows\system32\cmd.exe
0:\Users\takeshita>javac
使用方法: javac <options> <source files>
使用可能なオプションには次のものがあります。
-g                すべてのデバッグ情報を生成する
-g:none          デバッグ情報を生成しない
-g:{lines,vars,source} いくつかのデバッグ情報のみを生成する
-nowarn          警告を発生させない
-verbose         コンパイラの動作についてメッセージを出力する
-deprecation     非推奨のAPIが使用されているソースの場所を出力する
-classpath <path> ユーザー・クラス・ファイルおよび注釈プロセッサを検索する位置を指定する
-cp <path>       ユーザー・クラス・ファイルおよび注釈プロセッサを検索する位置を指定する
-sourcepath <path> 入力ソース・ファイルを検索する位置を指定する
-bootclasspath <path> ブートストラップ・クラス・パスの場所をオーバーライドする
-extdirs <dirs>   インストール済拡張機能の場所をオーバーライドする
-endorseddirs <dirs> 推奨規格パスの場所をオーバーライドする
```

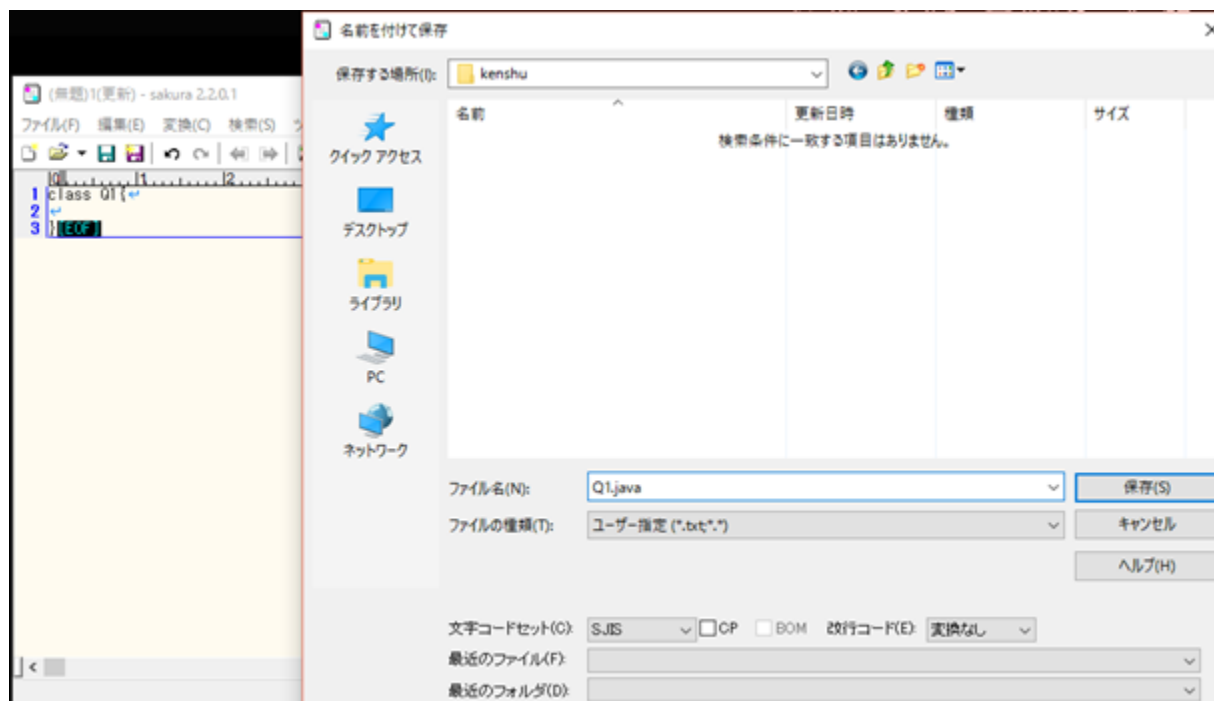
・Java実行手順

①サクラエディタ、秀丸等のテキストエディタを開き、Javaのソースコードを書く

これを任意の場所（例：デスクトップの自分のフォルダ等）に保存する

ファイルを保存するときの名前は、「クラス名.java」

例) ここではQ1というクラスを作成しているので、ファイル名は「Q1.java」とする



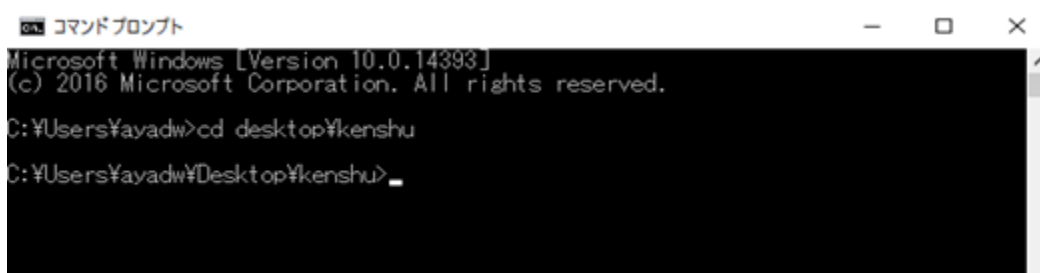
②次に、コマンドプロンプトでコンパイルを行う

コマンドプロンプトを開くと、このような画面が出てくるので、自分の作成したJavaファイルの置いてあるディレクトリに移動する



```
コマンドプロンプト
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\ayadw>
```

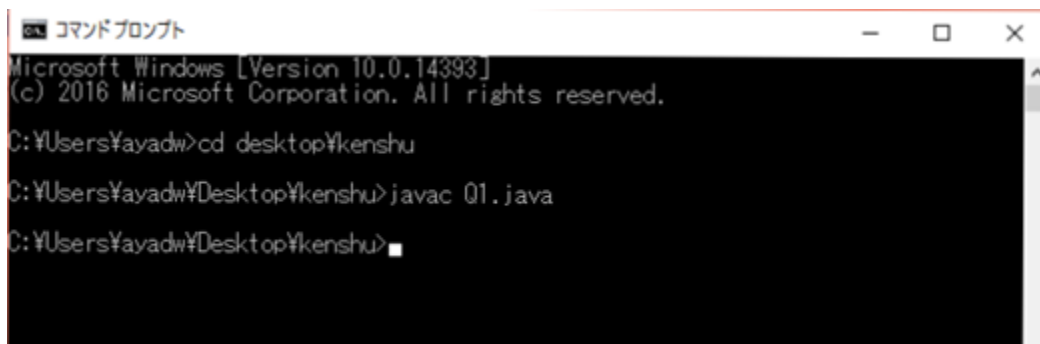
例) `cd desktop\kenshu` ※`cd`がディレクト移動のコマンド



```
コマンドプロンプト
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\ayadw>cd desktop\kenshu
C:\Users\ayadw\Desktop\kenshu>
```

ディレクトリの移動ができれば、Javaファイルのコンパイルを行う

例) `javac Q1.java` ※パスの設定ができていないと、`javac`コマンドが通らないので注意



```
コマンドプロンプト
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\ayadw>cd desktop\kenshu
C:\Users\ayadw\Desktop\kenshu>javac Q1.java
C:\Users\ayadw\Desktop\kenshu>
```

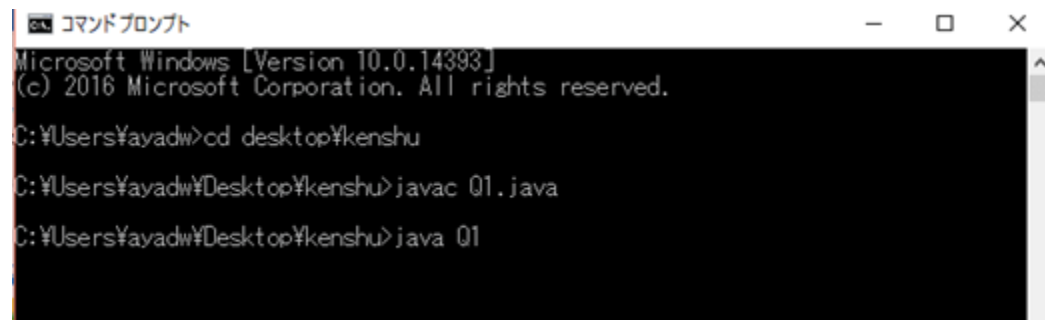
コンパイルを行うと、同ディレクトリ内にクラスファイルが作成される。

※`kenshu`フォルダにある`Q1.java`ファイルをコンパイルした場合、`研修`フォルダ内に`Q1.class`ファイルが作成されます。

③最後に実行を行う

コンパイルで何も表示されなかったらコンパイルが成功しているので、続いて実行を行う。

例) java Q1



```
コマンドプロンプト
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Yayadw>cd desktop\kenshu
C:\Users\Yayadw\Desktop\kenshu>javac Q1.java
C:\Users\Yayadw\Desktop\kenshu>java Q1
```

このような手順でJava基礎問題を進めていってください。

Java基礎課題

・問 1

「Hello World.」を画面に表示するJAVAファイルを作成して、それを実行せよ

例文：

```
public class Hoge { // hogeクラス
    public static void main (String[] args) { // メインメソッド
        System.out.println("こんにちは 世界"); // 処理
    }
}
```

解説：

- ・クラス名はファイル名と同一にしてください。
例) hogehoge.javaファイルの場合⇒「public class Hogehoge { 」になります。
- ・クラス名の先頭は大文字にしてください。
- ・クラスの中括弧({ })の中にメソッドがあり、メソッドの中に処理を書いていきます。
- ・「System.out.println」で文字を出力しています。

・問 2

キーボードで入力した整数nが偶数か奇数かを判定し、その結果を表示する。

また、表示する際の文言は画像を参照してください。

例文：

```
import java.io.*;

public class Hoge { // クラス
    public static void main (String[] args) throws IOException {
        // 入力処理の準備
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);

        // 入力した文字を入れるための変数を用意
        String inputData;

        // キーボード入力してもらい、作成した変数に入力値を格納
        inputData = br.readLine();

        // 条件によって出力する文字を変更する
        if ("AAA".equals(inputData)) {
            System.out.println("AAAを入力した！");
        } else {
            System.out.println("別の文字を入力した！");
        }
    }
}
```

解説：

- ・Stringやint型などの[変数](#)を使用しましょう。
- ・String型からint型への[型変換](#)を行きましょう。
- ・[if文](#)を使って条件分岐しましょう。
- ・[算術演算子](#)と[比較演算子](#)を学びましょう。

- ・ [文字列の比較](#)は「==」ではないので注意しましょう。

・ 問 3

キーボードで整数nを入力し、

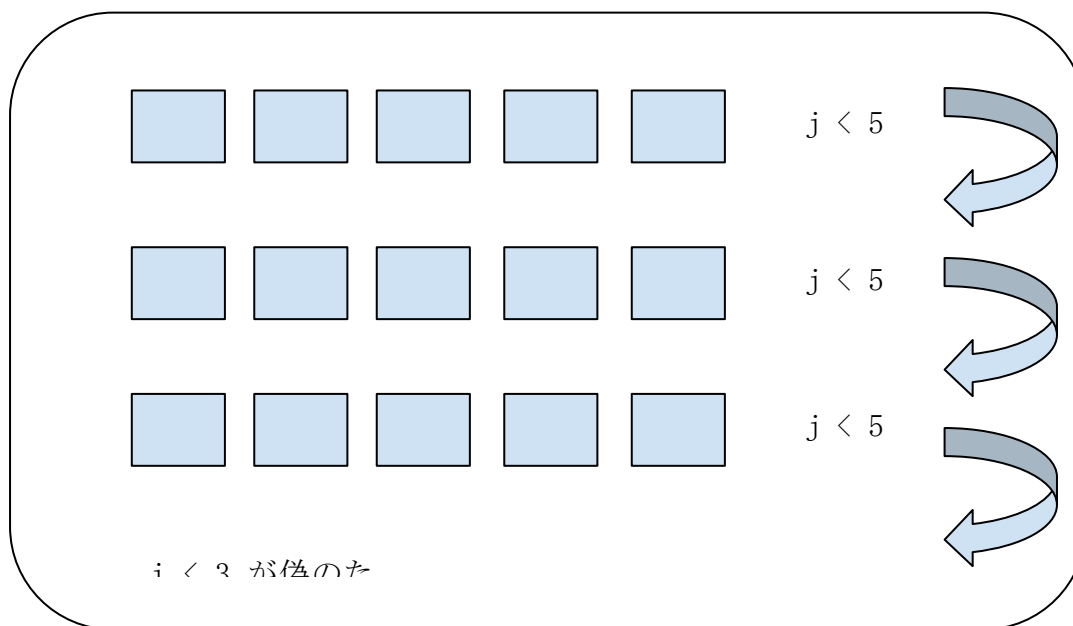
下の画像のように対角線の入った四角形(大きさは $n \times n$)を表示する。

例文：

```
public class Hoge {  
    public static void main (String[] args) throws IOException {  
        // 5 × 3 の長方形を出力する  
  
        // 行数分繰り返す  
        for ( int i = 0; i < 3 ; i++) {  
            // 列数分繰り返す  
            for (int j = 0; j < 5 ; j++) {  
                System.out.print("■");  
            }  
            // 列が終われば改行を行う  
            System.out.println("");  
        }  
    }  
}
```

解説：

- ・ [for文](#)を使いましょう。
- ・ 例文での変数iとjの動きのイメージは以下になります。



・ 問 4

キーボードで整数 n を入力し、 $100/n$ の結果を表示する。

その後、“終了”を表示すること

※下記exceptionをcatchした時エラー文を表示すること。

ArithmeticException：“0で除算できません”

NumberFormatException：“数字以外が入力されました。”

例文：

```
public class Hoge {
    public static void main (String[] args) {
        // 入力処理の準備
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);

        try {
            // 文字の入力
            String str = br.readLine();
            System.out.println(str);
        } catch ( IOException e) {
            System.out.println("文字列入力時に例外エラーが発生しました。");
            System.out.println("スタックトレース:" + e);
        }
    }
}
```

```
    } finally {  
        System.out.println("終了");  
    }  
}  
}
```

解説：

- ・ [try-catch文](#)を使いましょう。
- ・ 次課題以降はメソッドに記載している 「throws IOException」 を削除し、catchで記載するようにしましょう。「throws」 は呼び出し元に投げます。mainメソッドは最上位なのでコンソールに投げることになります。

・問5

キーボードで0～6を入力し、対応した曜日表示する。

例：0→日曜日～6→土曜日

switch文を使用すること。

例文：

```
public class Hoge {
    public static void main (String[] args) {
        // 入力処理の準備
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);

        try {
            // 文字の入力
            String str = br.readLine();
            switch (str) {
                case "A":
                    System.out.println("非常に不満");
                    break;
                case "B":
                    System.out.println("少し不満");
                    break;
                case "C":
                    System.out.println("どちらとも言えない");
                    break;
                case "D":
                    System.out.println("少し満足");
                    break;
                case "E":
                    System.out.println("大変満足");
                    break;
            }
        } catch (IOException e) {
            System.out.println("文字列入力時に例外エラーが発生しました。");
        }
    }
}
```

解説：

- ・ [switch文](#)を使いましょう。

・ 問 6

文字列を5つ入力し、それらを配列に入れてから再度並べて表示する。

例文：

```
public class Hoge {  
    public static void main (String[] args) {  
        // 配列を定義する。  
        int[] intArray = {1, 2, 3, 4, 5};  
  
        // 表示する  
        for ( int i = 0; i < intArray.length; i++){  
            System.out.println(intArray[i]);  
        }  
    }  
}
```

解説：

- ・ [配列](#)の定義の仕方、値の取り方を学びましょう。
- ・ lengthを使うことにより、配列の要素数が変わっても条件式の修正が不要になります。

・問 7

整数を5つ入力してそれらを昇順に並べ替えて、並び替えの結果と最大値、最小値を表示する。

※arrays.sortは使用しないこと

例文：

```
public class Hoge {
    public static void main (String[] args) {
        // 配列を定義する。
        int[] intArray = {4, 2, 5, 1, 3};
        // 並び替え用変数
        int wk = 0;

        // 並び替えを行う
        for ( int i = 0; i < intArray.length; i++) {
            for ( int j = intArray.length-1 ; j > i ; j-- ) {
                // 値の比較
                if ( intArray[j - 1] >= intArray[j]) {
                    // 値の入れ替え
                    wk = intArray[j - 1];
                    intArray[j - 1] = intArray[j];
                    intArray[j] = wk;
                }
            }
        }
    }
}
```

解説：

- ・ [ソートアルゴリズム](#)を学び、使用しましょう。
- ・ 昇順、降順を間違えないようにしましょう。

・問 8

設問3と同様のキーボードから整数 n を入力し、

以下のような対角線の入った四角形(大きさは $n \times n$)を書く。

※ただし main メソッド内は2行以内にする

※ヒント：同じクラス内に別メソッドを作成する

例文：

```
public class Hoge {  
    public static void main (String[] args) {  
        printHello();  
    }  
  
    public void printHello() {  
        System.out.println("Hello!!");  
    }  
}
```

解説：

- ・ [メソッド](#)について学びましょう。

・問 9

キーボードで整数を2つ入力し、入力した数字の積とそれを2で割った数を求める。

※入力値を引数とし計算結果を戻り値として返すメソッドとしてそれぞれ作成すること

例文 :

```
public class Hoge {
    public static void main (String[] args) {
        int num1 = 5;
        int num2 = 3;

        System.out.println("答えは：" + addition(num1, num2));
    }

    public int addition(int a , int b) {
        int result = a + b;

        return result;
    }
}
```

解説 :

- ・ [引数と戻り値](#)について学びましょう

・問 1 0

追加したコンストラクタを利用してインスタンスごとに名前が変更されるようする。

例文：

```
public class Hoge {
    public static void main (String[] args) {
        // 名前の設定を行わずインスタンスを生成
        SelfIntroduction si = new SelfIntroduction();
        si.printName();

        // 名前を設定したインスタンスを生成
        SelfIntroduction si2 = new SelfIntroduction("兵藤和尊");
        si2.printName();
    }
}

public class SelfIntroduction() {
    private String name;

    public SelfIntroduction() {
        this.name = "伊藤開司";
    }

    public SelfIntroduction(String data) {
        this.name = data;
    }

    public void printName() {
        System.out.println("私の名前は " + this.name + " です");
    }
}
```

解説：

- ・ [インスタンス](#)について学びましょう
- ・ [コンストラクタ](#)を使用しましょう。
- ・ [オーバーロード](#)を学びましょう



・問 1 1

継承機能を使用して子クラスと、メインのメソッド用のクラスを作成し、画像通りに表示すること

例：

```
class Student { // 親クラス
    String name;
    int id;
    protected Student(String param1, int param2) {
        this.name = param1;
        this.id = param2;
    }
    protected void printName() {
        System.out.println("生徒番号"+ id + "は、" + name + "です。");
    }
}

public class Score extends Student { // 子クラス
    int kokugoScore;
    public Score(String param1, int param2, int score1) {
        super(param1, param2);
        this.kokugoScore = score1;
    }
    public void printName() { // オーバーライド
        super.printName();
        System.out.println("国語は"+kokugoScore+"点です。");
    }
}

public class Main { // メインクラス
    public static void main(String[] args) {
        Score score = new Score("田中将大", 100, 25);
        score.printName();
    }
}
```

解説：

- ・ [継承](#)について学びましょう
- ・ メソッドの [オーバーライド](#)をしましょう

・問 1 2

設問9と同様にキーボードで整数を2つ入力し、

入力した数字の積とそれを2で割った数を求める。

※以下のインタフェースを実装して入力した数字の積、それを2で割る計算を行うクラスを作成すること

例：

```
interface Calc {
    int addition(int a, int b);

    int subtraction(int a, int b);
}
// 実装クラス
class AddSub implements Calc {
    public int addition(int a, int b) {
        return a + b;
    }
    public int subtraction(int a, int b) {
        return a - b;
    }
}

public class Main {
    public static void main(String[] args) {
        // インスタンス生成
        Calc calc = new AddSub();

        System.out.println(calc.addition(10, 5));
        System.out.println(calc.subtraction(10, 5));

    }
}
```

解説：

- ・ [インターフェース](#)について学びましょう

・問 1 3

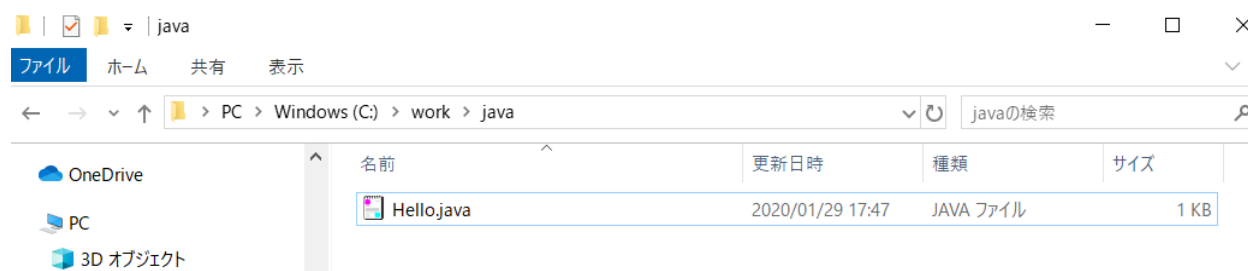
設問1と同様に「Hello World!!」を画面に表示する。

※作成するクラスのパッケージは「kenshu.wytel」とすること

例：

```
package lang.java.hello;

public class Hello {
    public Hello() {
        System.out.println("Hello Java!");
    }
}
```



```
C:\>java work/java/Hello
Hello Java!

C:\>
C:\>
C:\>
```

解説：

- ・パッケージとはクラスやインターフェースをフォルダでグループ分けして格納することです。
- ・パッケージ用のフォルダを作成しましょう。

・問 1 4

入力を2回行い、1回目と2回目の文字が同じかどうか表示する。

1回目と2回目の文字が異なる場合はもう一度入力を促す。

1回目の文字が空文字はもう一度入力を促す。

2回目の文字が空文字はもう一度入力を促す。

例：

```
public class hoge {  
    public static void main (String[] args) {  
        int i = 0;  
        // 無限ループ  
        while(true) {  
            i++;  
            // 偶数でない場合再ループ  
            if(i % 2 != 0) {  
                continue;  
            }  
            // 出力  
            System.out.println(i);  
            // 20を超えたら終了  
            if(i > 20) {  
                break;  
            }  
        }  
    }  
}
```

解説：

- ・ [while](#)文を学びましょう。
- ・ [ループからの抜け方](#)を学びましょう。
- ・ [NULLと空文字の違い](#)を理解しましょう。

・問 1 5

設定したい数だけ値を入力して、これらの平均を表示する。

例：平均を表示したい値を入力してください。

平均を表示したい値をすべて入力したら「OK」を入力してください。

例：

```
import java.util.ArrayList;
import java.util.List;

public class hoge {
    public static void main (String[] args) {
        List<Integer> list = new ArrayList<Integer>();

        for(int i = 0; i < 23 ; i++) {
            list.add(i);
        }

        // 合計
        int result =0;
        for(int j = 0; j < list.size() ; j++) {
            result += list.get(j);
        }

        System.out.println(result);
    }
}
```

解説：

- ・ [ArrayListクラス](#)を使いましょう
- ・ 配列とarrayListの違いを説明できるように理解しましょう。

Java基礎知識

・ Javaとは

Javaとはプログラミング言語の1種で、OSなどのプラットフォームに依存しないため、様々な現場で多く使われている言語です。他にはC#やPythonなど、様々な言語があるので興味があれば調べてみてください。

・ コンパイル

Javaではプログラムを動かす前にコンパイルを行います。

拡張子「. java」 ファイルに記載したコードをコンピュータが認識できる機械語に変換して「. class」ファイルを作成しているイメージです。

javaファイルを編集後、コンパイルを行わないと実行時に反映されないので気を付けてください。

・ 変数

変数とは、簡単に言うと値を入れる箱です。

箱には型があり、基本的に型にあった値を箱に入れます。

変数の定義、使い方は以下を参考にしてください。

```
// 変数の定義
int i ;

// 変数に初期値を設定
i = 100;

// 変数の定義と初期値の設定を一緒に行えます。
int data = 0;
```


・ 型

変数を定義する際の型は以下表の種類をよく使います。

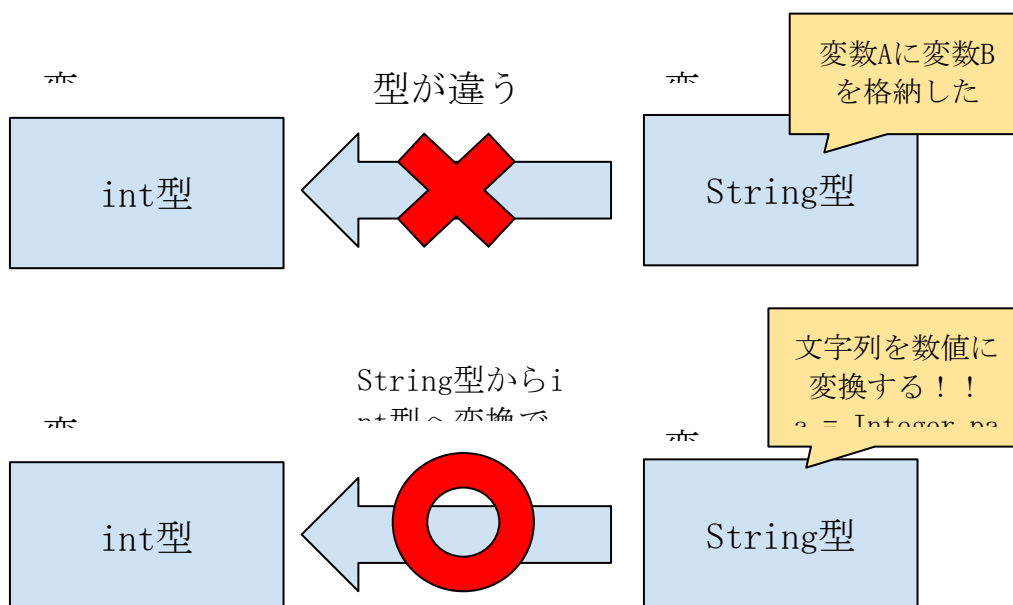
分類については調べてみてください。

これらは一部なので他にも型の種類はあります。

型	値	分類
int	数値	基本データ型（プリミティブ）
String	文字列	参照型
boolean	true / false	基本データ型（プリミティブ）
オブジェクト (クラス)	インスタンス化する独自クラスなど	参照型

・ 型変換

変数の型が違うときに値の型変換を行い、値を格納したい場合があります。そんなときは型変換を行って変数に値を格納することが可能です。



・ if 文

プログラムをしていくうえで、以下のような場面があると思います。

例1：int型の変数numの値が1の時、処理Aを行う。

変数numの値が2の時、処理Bを行う。

変数numの値が上記以外の場合は処理Cを行う。

このような場面に、if文を使うことで条件による分岐が可能になります。

ifの括弧の中に条件式を書き、条件式の結果が真(true)の場合に処理Aを行います。

elseは、条件の結果が偽(false)の場合に処理Bを行います。

```
if (条件式) {  
    処理A           // 条件式が真の処理  
} else {  
    処理B           // 条件式が偽の処理  
}
```

else ifを記載すれば、条件式Aが偽の場合は条件Bの判定が出来ます。

```
if (条件式A) {  
    処理A  
} else if (条件式B) { // 条件Aが偽なので条件Bの判定を行う  
    処理B  
} else { // 条件A、条件Bどちらも偽の場合の処理  
    処理C  
}
```

上記の例1をif文にすると以下のような形になります。

```
if ( num == 1 ) {           // numの値が1の場合  
    System.out.println("処理Aです!"); // 処理Aを行う  
} else if ( num == 2 ) {   // numの値が2の場合  
    System.out.println("処理Bです!"); // 処理Bを行う  
} else {                   // numの値が1でも2でもない場合
```

```
System.out.println("処理Cです!"); // 処理Cを行う
}
```

・演算子とは

計算でよく使う「+」や「-」や「=」などがjavaにはたくさん用意されており、それらを演算子と言います。

javaでは主な演算子の分類として、算術演算子、比較演算子、代入演算子などがあります。

本研修では代入演算子は触れませんが、興味があれば調べてみてください。

・算術演算子

算術演算子には、四則演算子とインクリメント演算子があります。

主に使うものは以下一覧です。

演算子	概要
+	加算 or 文字列の結合
-	減算
*	乗算
/	除算
%	余り
++	インクリメント(+1する)
--	デクリメント(-1する)

使用例：

```
int a = 13;
int b = 5;
int ans = 0; //結果格納用

ans = a + b ;    // ansは 18
ans = a - b ;    // ansは 8
ans = a * b ;    // ansは 65
ans = a / b ;    // ansは 2
ans = a % b ;    // ansは 3
```

```
ans = ++b;      // ansは 6※
ans = b++;      // ansは 5※
ans = --a ;     // ansは12※
ans = a--;      // ansは13※
```

※前につけると計算後に格納、後ろにつけると格納してから計算なので注意

・比較演算子

比較演算子は主に以下の演算子を使います。

演算子	読み	書き方	概要
>	大なり	a > b	aはbより大きい
>=	大なりイコール	a >= b	aはb以上
<	小なり	a < b	aはbよりも小さい
<=	小なりイコール	a <= b	aはb以下
==	イコール	a == b	aとbは等しい
!=	ノットイコール	a != b	aとbは等しくない

・論理演算子

複数の条件の結果が真偽かの判定を行うための演算子になります。

主に以下の演算子を使います。

演算子	使い方	概要
&&	条件1 && 条件2	条件1が真かつ条件2が真の時、真となる
	条件1 条件2	条件1が真または条件2が真の時、真となる

使用例：

```
if ( i == 1 && j == 1) {
    system.out.println("iとjどちらも1だよ！");
}

if ( i == 1 || j == 1) {
    system.out.println("iとjのどちらかが1だよ！");
}
```



・文字列の比較

文字列が一致しているか比較の場合、条件式「==」で判定することはできません。

Stringはクラスなのでメソッドを使用して比較を行いましょう。

例：

```
String str = "AAA";
if("AAA".equals(str) {
    System.out.println("文字列が一致しました。");
}
```

・for文

for文は繰り返し処理を行い時に利用します。

初期値・終了条件・計算式がまとめて記載出来るのが特徴です。

終了条件が真の時は繰り返し、偽になったときにfor文から出ます。

```
for ( 初期値の定義 ; 終了条件 ; 計算 ) {
    繰り返し行う処理
}
```

使用例：

```
for (int i = 0; i < 5 ; i++) {
    system.out.println(i);
}
```

----出力結果----

```
0
1
2
3
4
```

拡張for文という書き方がありますが、本研修では触れないと思うので興味があれば調べてみてください。



・ try-catch文

プログラムを実行するうえで、様々な例外エラーが発生します。(例：0除算など)

例外が発生しうる箇所にtry-catchを記載して、どのような例外エラーが発生しているか、どこで発生しているかを把握することが出来ます。

```
try {
    例外が発生する可能性のある処理
} catch( 発生しうる例外のクラス 変数 ) {
    例外をキャッチしたときの処理(エラーメッセージを出す等)
} finally {
    例外が発生しててもしてなくても必ず行う
}
```

使用例：

```
try {
    // nullの変数の機能を使おうとする。
    String str = null;

    if(str.equals("a")){
        system.out.println("aだよ！");
    }

} catch (NullPointerException e) {
    System.out.println("ぬるぽ");
} finally {
    System.out.println("処理を終了します。");
}
```


・Switch文

switch文は、式の結果の値に一致するcaseの位置へ処理を移動して条件分岐を行う方法です。
一致しない場合はdefaultの中の処理を行います。

```
switch (式) {  
    case 定数:  
        処理1  
        break;  
    case 定数:  
        処理2  
        break;  
} default {  
    一致しなかった時の処理  
}
```

注意点

- ・ char, byte, short, int, Stringの型が使用可能です。
- ・ breakを記載しないと後続の処理も行われます。

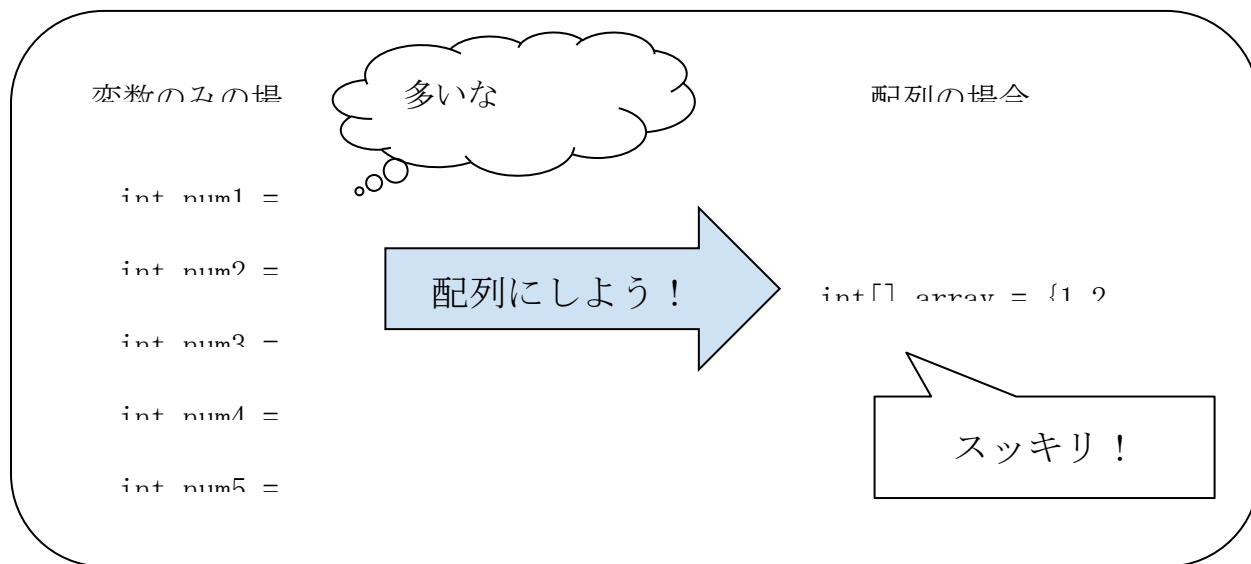
使用例：

```
int num = 1;  
switch (num) {  
    case 1:  
        System.out.println("A");  
        break;  
    case 2:  
        System.out.println("B");  
        break;  
} default {  
    System.out.println("一致なし");  
    break;  
}  
  
----出力----  
A
```

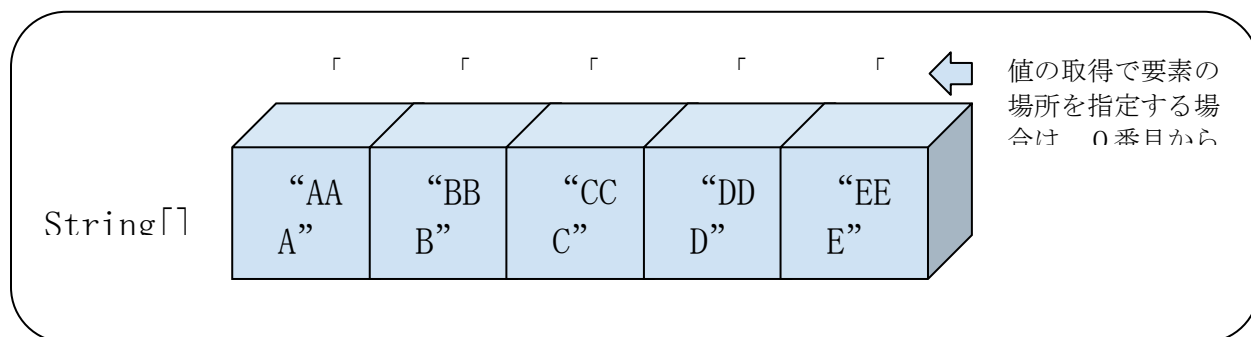
```
// breakがない場合  
int num = 1;  
switch (num) {  
    case 1:  
        System.out.println("A");  
    case 2:  
        System.out.println("B");  
} default {  
    System.out.println("一致なし");  
}  
  
----出力----  
A  
B  
一致なし
```

・ 配列

配列とは、複数のデータを一つの変数にまとめたものです。



配列イメージ：



構文：

```
// 定義
型[] 変数名;

// 要素数を指定して定義
型[] 変数名 = new 型 [ 要素数 ];

// 要素の初期値を設定して定義
型[] 変数名 = {値1, 値2, 値3, ...};

// 配列の要素数を取得する
変数名.length
```

使用例：

```
String[] strArray = {"aaa", "bbb", "ccc", "ddd", "eee"};

System.out.println( strArray[4] );
System.out.println( strArray[0] );
System.out.println( strArray.length ); // 配列の要素数

----出力結果----
eee
aaa
5
```

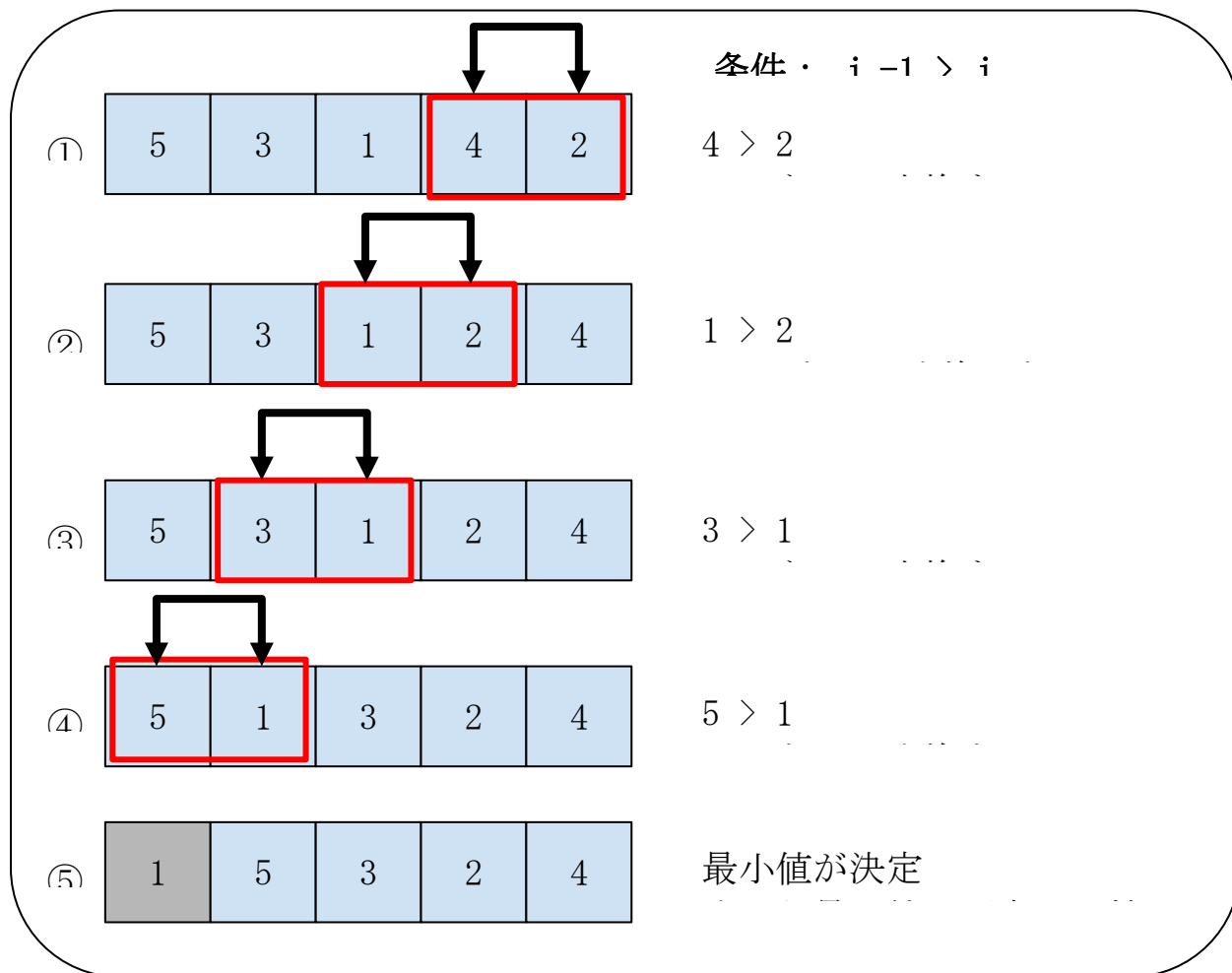
・ソートアルゴリズム

要素を昇順や降順に並び変える方法は様々なやり方があります。

ここでは比較的簡単なアルゴリズムを紹介します。

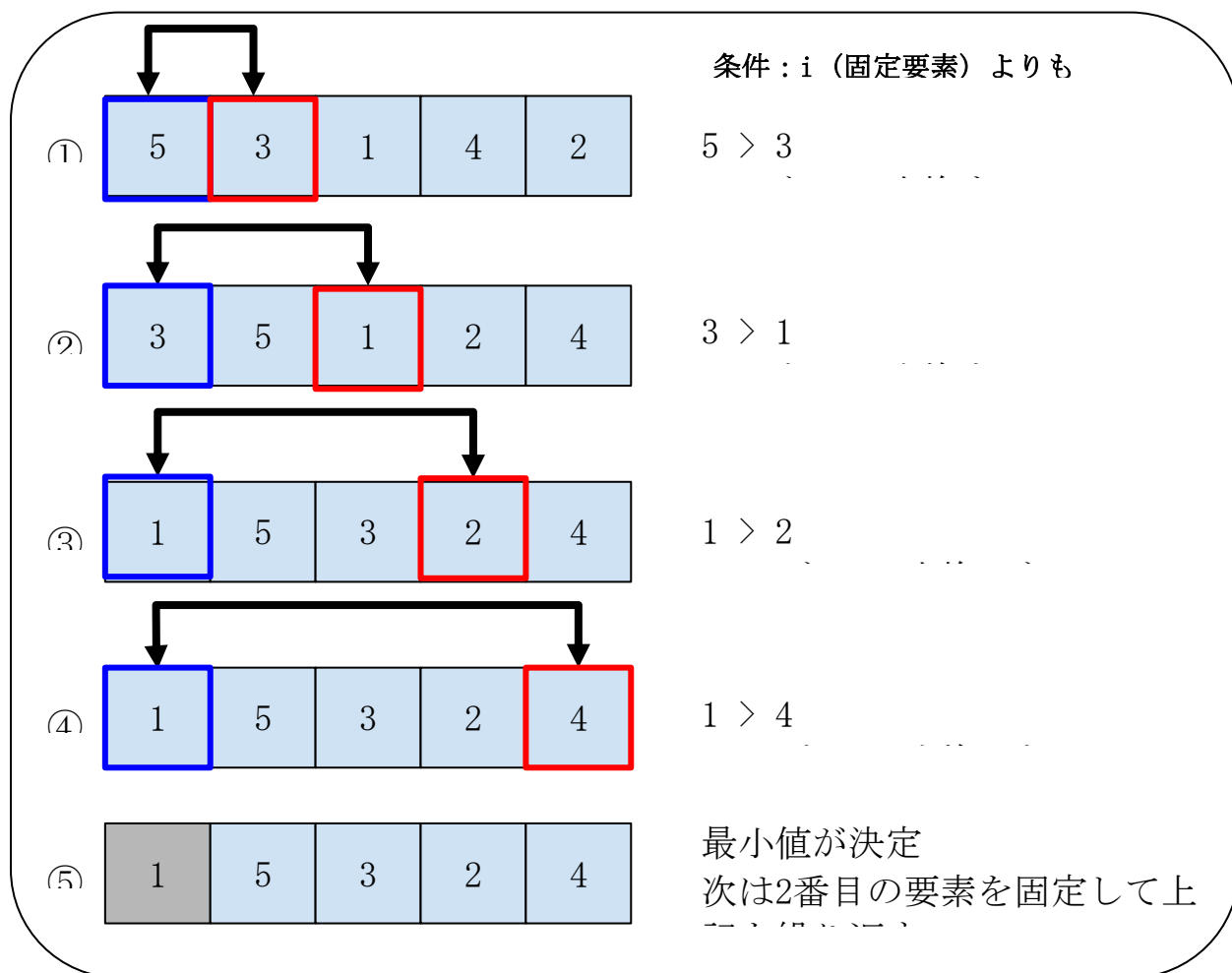
・バブルソート

隣り合う要素を見比べ判定を行い、値の入れ替えを繰り返していく手法です。



・選択ソート

選択ソートは、要素を固定して各要素と比較して入れ替えを行います。



・メソッド

メソッドとは、処理をひとまとめにしたものです。

メソッドを定義することによって、処理の再利用や編集が容易になる利点があります。

・メソッドの定義

```
アクセス修飾子 戻り値の型 メソッド名 (引数) {
    処理
}
```

引数を使用しない場合は省略可能です。

※[アクセス修飾子](#)、[引数と戻り値](#)については別途記載します。

例①：

```
public int addNum ( int a,int b) {
    int result = a + b;
    return result;
}
```

・メソッドの呼び出し

メソッドの呼び出し方は以下の通りとなります。

```
// 戻り値があるメソッドの場合、値を格納する変数を用意してください。
// 同クラス内呼び出し
戻り値を格納する型 変数名 = this.メソッド名 (引数) ; // thisは省略可能
```

上記例①を呼び出す処理を実際書いてみます。

例②：

```
// 同クラス内呼び出し
int addResult = addNum(10,5);
System.out.println( "足し算の答えは：" + addResult );

// 戻り値を変数に格納しないで直接出力することも可能です。
System.out.println( "足し算の答えは：" + addNum(2,5) );
```

インスタンスの呼び出しは以下の通りです。

```
// インスタンスから呼び出し
クラス名 インスタンス変数名 = new クラス名();
戻り値を格納する型 変数名 = インスタンス変数名.メソッド名 (引数);
```

例③：

```
// インスタンスから呼び出し
Calculation calc = new Calculation();
int result = calc.addNum(10, 5);
System.out.println(addResult);

// 呼び出しクラス
public class Calculation {
    public int addNum(int a, int b) {
        int result = a + b;
        return result;
    }
}
```

・アクセス修飾子

アクセス修飾子とは、フィールドやメソッドの参照および実行できる範囲を設定するための物です。

設定することでカプセル化を行い、むやみに値の変更などを行えないように制御できます。

以下のアクセス修飾子を設定することが出来ます。

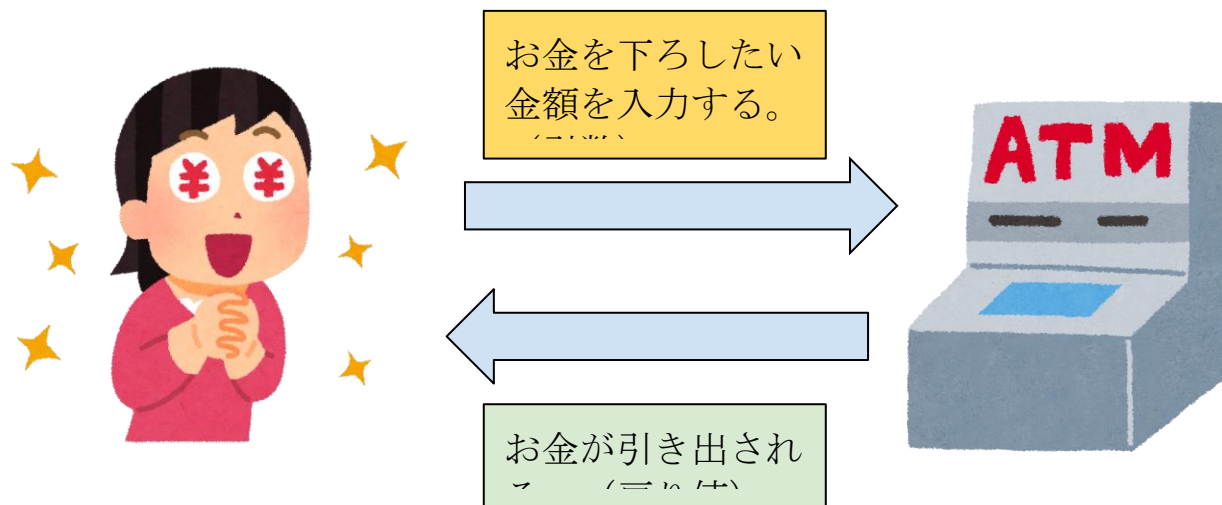
アクセス修飾子	概要
public	すべてのクラスからアクセス可能
protected	現在のクラスとサブクラスからアクセス可能
なし	現在のクラスと同じパッケージのクラスからアクセス可能
private	現在のクラスからだけアクセス可能

・引数と戻り値

引数とは、メソッドが処理を行う上で必要な値を渡すことです。

戻り値とは、メソッドが処理を行った結果を呼び出し元へ返却することです。

ATMを例にイメージしてみましょう。



引数は複数定義できます。複数定義する際はカンマで区切りましょう。

引数、戻り値共にクラスなどの参照型変数も定義することが出来ます。

例：

```
public int addNum ( int a,int b) {  
    int result = a + b;  
    return result;  
}
```

引数を必要としない場合は、省略できます。

戻り値がない場合は、「void」を記載します。

また、returnも必要ないです。

例：

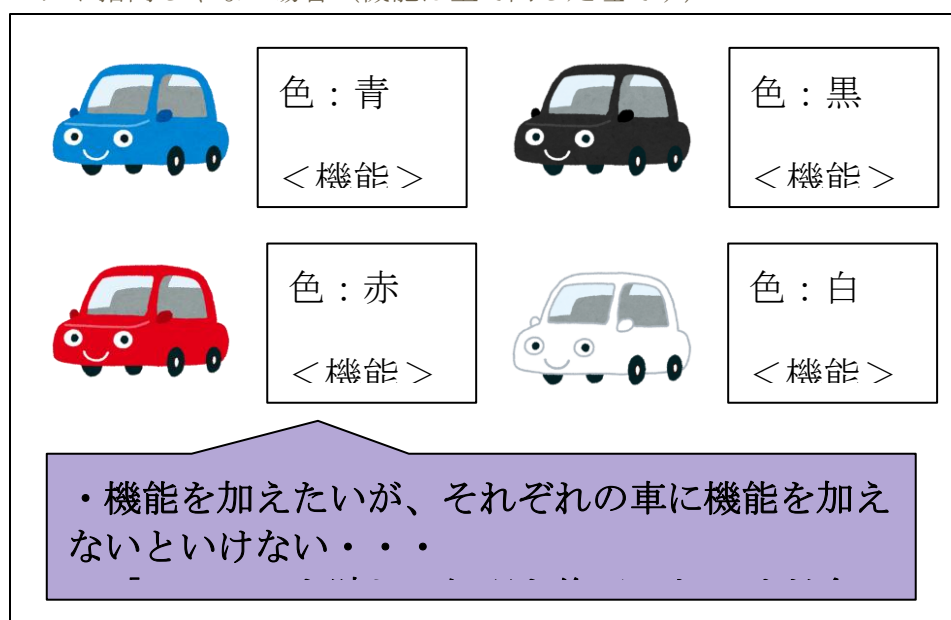
```
public void printHello() {  
    System.out.println("Hello!");  
}
```


・オブジェクト指向

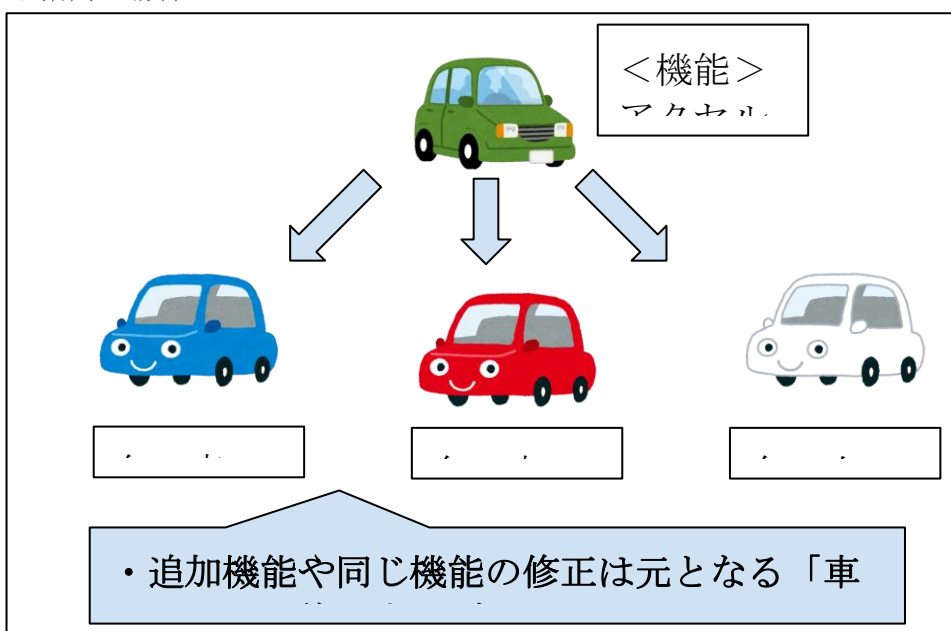
オブジェクト指向は、プログラムを作成するときの「考え方」「概念」です。
そのため明確に「これが答えだ」という定義が難しいので、研修を行う内になんとか理解していければと思います。

クラスを物体（オブジェクト）として捉えて、クラスの共通化や部品化を行い効率よく作業が行えることができます。

例：オブジェクト指向じゃない場合（機能は全て同じ処理です）



オブジェクト指向の場合



・インスタンス

インスタンスとは、クラスをもとに生成するオブジェクトのこと

別クラスのフィールドやメソッドを使うためにオブジェクトとして生成して使用することが出来ます。

例：

```
public class Person{
    private String name;
    // 名前を設定する。
    public void setName(String data) {
        this.name = data;
    }
    // 自己紹介する。
    public void printName() {
        System.out.println("my name is " + this.name);
    }
}

public class Main{
    public static void main(String[] args) {
        // Personクラスをインスタンス化
        Person person1 = new Person();
        person1.setName("Bon");

        Person person2 = new Person();
        person2.setName("Jovi");

        person1.printName();
        person2.printName();
    }
}
```

----出力----

my name is Bon

my name is Jovi



・コンストラクタ

コンストラクタとはインスタンス化したときに呼び出されるメソッドです。

主に初期値の設定に用いられます。

オーバーロードをすることで引数による初期値の設定が変更できます。

例：

```
// インスタンス化するクラス
public class Person {
    private String name;
    private int hp;
    // 引数なしの場合
    public Person() {
        this.name = "名無し";
        this.hp = 100;
    }
    // 引数で指定した値を設定(オーバーロード)
    public Person(String nameData, int hpData) {
        this.name = nameData;
        this.hp = hpData;
    }
    public void printData() {
        System.out.println("勇者" + this.name + " HP:" + this.hp);
    }
}

// メインクラス
public class Main {
    public static void main(String[] args) {
        // インスタンス化
        Person yuusya1 = new Person();
        Person yuusya2 = new Person("ああああ", 50);

        yuusya1.printData();
        yuusya2.printData();
    }
}
```

----出力----

勇者名無し HP:100
勇者ああああ HP:50

・オーバーロード

オーバーロードとは、同じクラスの中でメソッド名が同じで、引数の型や数、並び順が違うメソッドを2つ以上定義することをいいます。

同じような処理だけれども、一部を変数にして処理結果をより自由に変えたい場合などで使うと便利です。

以下の例だとコンストラクタのPersonメソッドの引数の数を変更してオーバーロードを行っています。

例：

```
// インスタンス化するクラス
public class Person {
    private String name;
    private int hp;

    // 引数なしの場合
    public Person() {
        this.name = "名無し";
        this.hp = 100;
    }

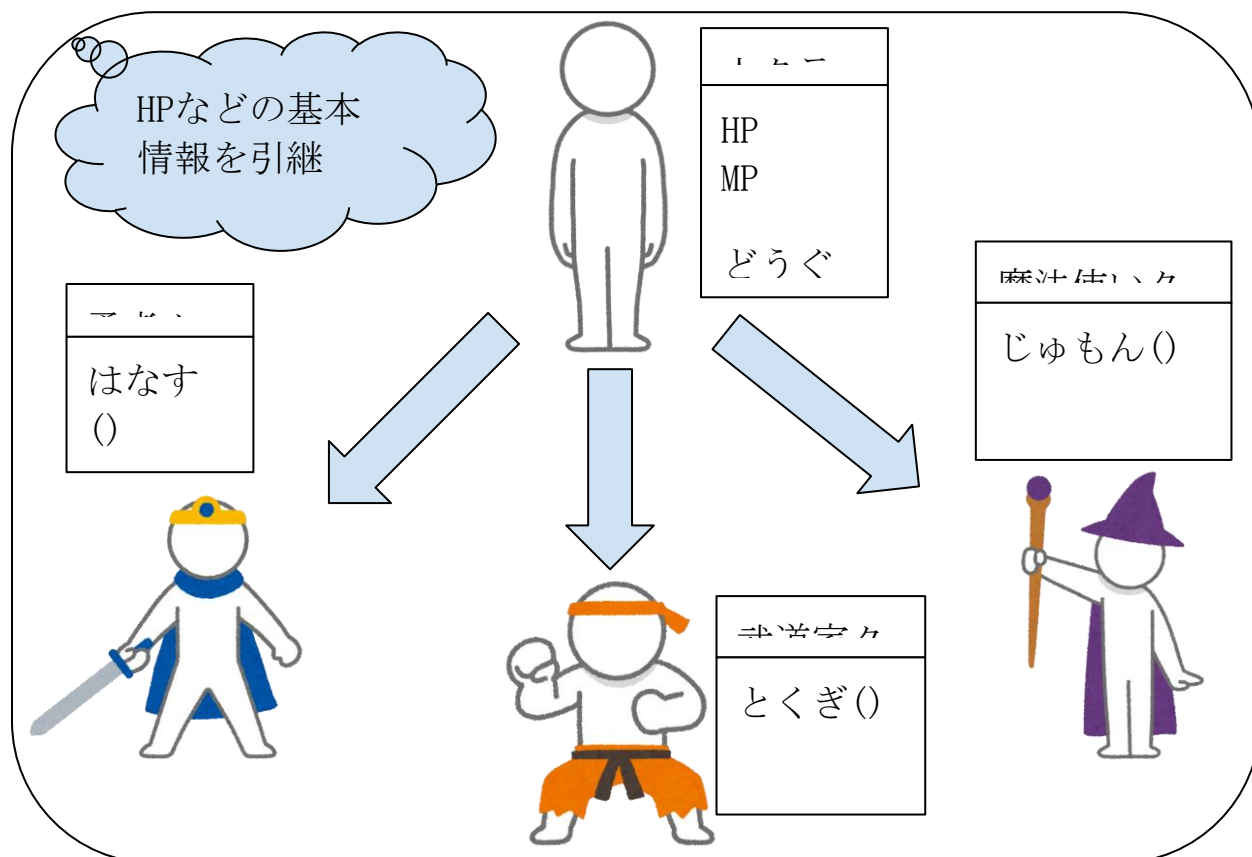
    // 引数で指定した値を設定
    public Person(String nameData, int hpData) {
        this.name = nameData;
        this.hp = hpData;
    }

    public void printData() {
        System.out.println("勇者" + this.name + " HP:" + this.hp);
    }
}
```

・ 継承

継承とは、親クラスのフィールドやメソッドを子クラスに継承して、子クラスに独自のメソッドなどを作成する手法です。

継承のイメージをドラクエのパーティでやってみましょう。



親クラスを継承して子クラスを定義するときは、「extends」を記載します。

```
public class Hero extends Person {
    . . .
}
```

子クラスから親クラスを参照する場合は、「super」を使います。

```
// 親クラスフィールド参照
super.HP = 100;
// 親クラスメソッド実行
super.useItem();
// 親クラスコンストラクタ呼び出し
super();
```

・オーバーライド

オーバーライドとは、「継承した親クラスのメソッドの内容を子クラスで再定義すること」です。

■オーバーライドとみなされる条件

- ・ 戻り値の型、メソッド名、引数の型、数、順番が同じであること。
- ・ アクセスレベルが親クラスと同じか緩い制限であること。
(例：親クラスがprotectedなら子クラスはpublicかprotectedを指定可能)

■オーバーライド出来ない条件

- ・ 「final」で定義されたメソッド
- ・ 「static」で定義されたメソッド

例：コンストラクタをオーバーライド

```
// 親クラス
public class Person {
    private String name;
    private int hp;
    private int mp;
    // コンストラクタ
    public Person() {
        this.name = "名無し";
        this.hp = 100;
        this.mp = 100;
    }
    public Person(String nameData, int hpData, int mpData) {
        this.name = nameData;
        this.hp = hpData;
        this.mp = mpData;
    }
}

// 子クラス
public class Hero extends Person {
    // オーバーライド
    public Person() {
        name = "勇者";
        hp = 200;
    }
}
```

```

    mp = 80;
  }
}

```

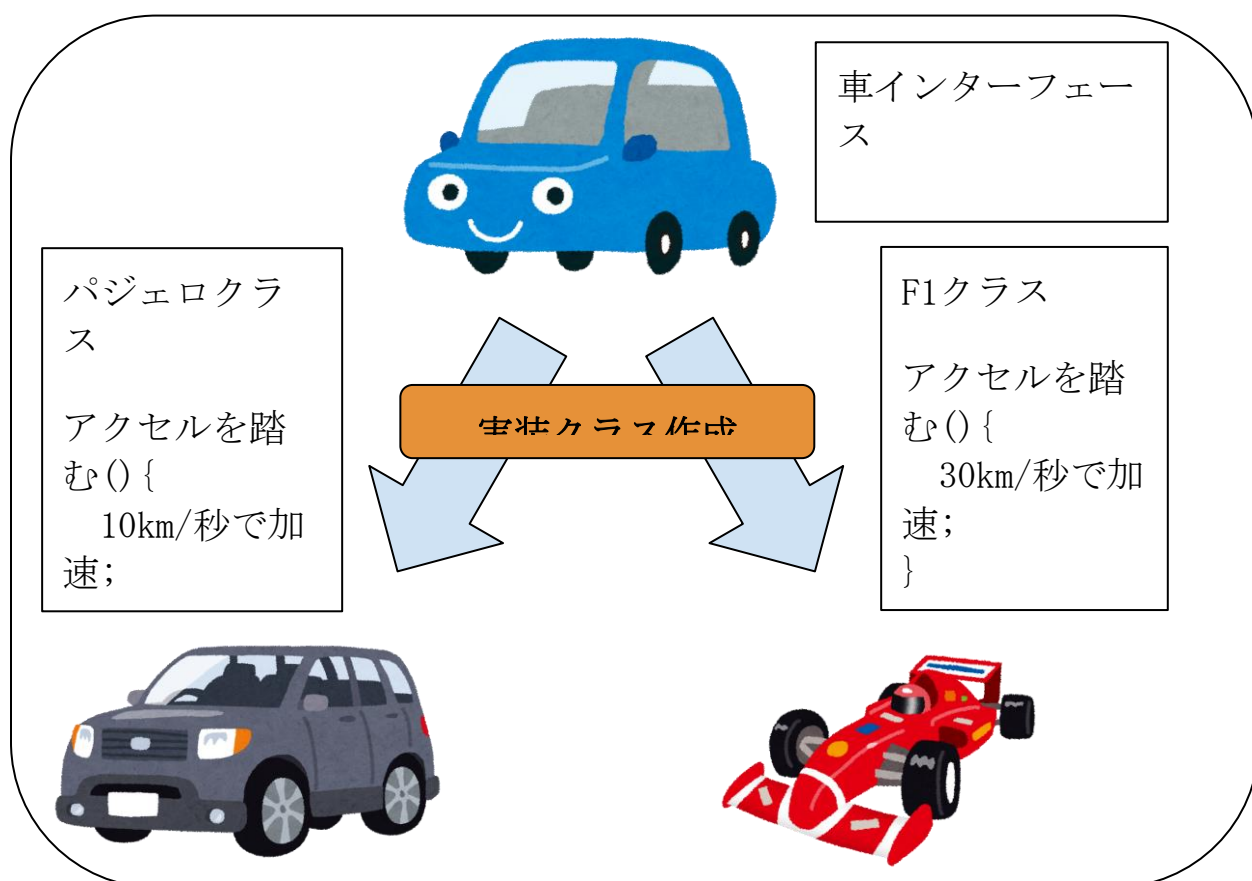
・インターフェース

インターフェースとは、フィールドやメソッドの仕様（戻り値とメソッド名、引数）のみを定義したものです。

インターフェースで最低限必要なものをルールのように定め、そのインターフェースを利用した実装クラスに実際の処理などを記載していきます。

また、複数のインターフェースを利用した多重継承がjavaでは許されています。

以下の例では、車というインターフェースから様々な車種を作成しています。



実装クラス作成時には、クラス名の後ろに「implements」を付けます。

```

public class F1クラス implements 車 {
    public void アクセルを踏む() {

```

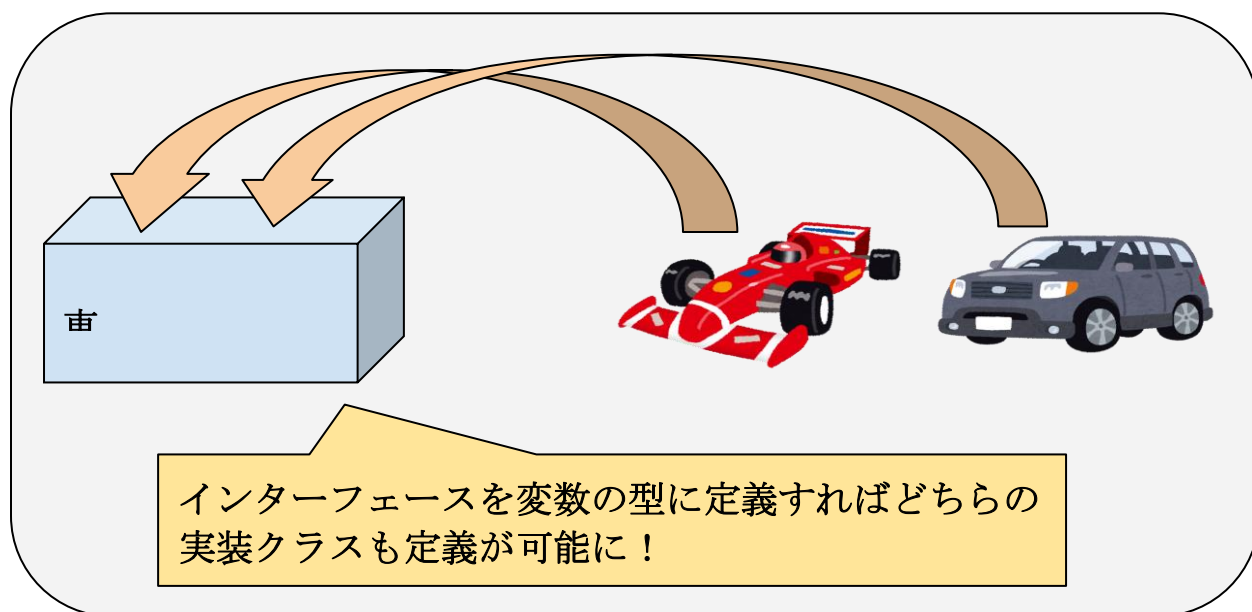


```

    . . .
}
public void ブレーキを踏む() {
    . . .
}
}

```

実装クラスをインスタンス生成するとき、インターフェースの型で受け取ることが出来るため、共通化して使用出来ます。



以下のように定義して使用します。

```

車 car = new F1クラス();
車 car2 = new パジェロクラス();

```

上記のようにインターフェースを使用して変数を定義すれば、条件によって使用する実装クラスの変更も可能になります。

・while文

whileは、繰り返し処理を行い時に利用します。

for文を同じ繰り返しですが、条件式のみで判定を行います。

終了条件が真の時は繰り返し、偽になったときにwhile文から出ます。

```
while( 終了条件 ) {  
    繰り返し行う処理  
}
```

使用例：

```
int i = 0;  
while (i < 5) {  
    system.out.println(i);  
}
```

----出力結果----

```
0  
1  
2  
3  
4
```

また、条件式に「true」を入れると無限ループになります。

・ループからの抜け方

for文やwhile文のループから強制的に抜けたいときに以下の処理で抜けることができます。

```
break;
```

再度ループさせるときは

```
continue;
```

例：

```
int i = 0;

// 無限ループ
while(true) {
    i++;

    // 偶数でない場合再ループ
    if(i % 2 != 0) {
        continue;
    }

    // 出力
    System.out.println(i);

    // 20を超えたら終了
    if(i > 20) {
        break;
    }
}
```

・NULLと空文字

Stringのnullと空文字はどちらも値が無いように見えますが違いがあります。

	概要
NULL	値の参照先や要素の定義がない状態です。 要素そのものが無いイメージです。
空文字（ “” ）	文字列要素はありますが、長さが0の文字です。



• ArrayListクラス

Listインターフェースの実装クラスとしてArrayListクラスがあります。

ArrayListクラスは要素数が決まっていない配列と思ってください。

「条件が一致している要素のみ配列に格納したい」など要素数が決まらない時に使用すると良いでしょう。

以下のメソッドを良く使われます。他にもいろいろあるので調べてみてください。

メソッド	概要
add(引数)	引数で渡した要素を一番後ろに格納する。
get(要素番号)	要素番号で指定した要素を取得します。
size()	要素数を出力します。

例：

```
List<Integer> list = new ArrayList<Integer>();

for(int i = 0; i < 20 ; i++) {
    // 乱数取得
    int random = (int)(Math.random() * 10);
    // 乱数の値が8以上になった値を格納
    if(random > 7) {
        list.add(i);
    }
}

// 乱数が
for(int j = 0; j < list.size() ; j++) {
    System.out.println("7を超えた要素番号:" + list.get(j));
}
```

・Static修飾子

変数やメソッドにStatic修飾子を付けると、静的(クラス)変数・静的(クラス)メソッドになります。

呼び出し方は、

```
クラス名. 変数名;  
クラス名. メソッド名();
```

static修飾子がついている変数・メソッドはインスタンスから参照することが出来ません。

インスタンス化していないため、1つの変数を共通で使用するイメージです。

そのため、1つのstatic変数を複数で参照している場合は、値の変更が他に呼び出している箇所にも影響するため、注意が必要です。

・修飾子一覧

修飾子		説明
アクセス 修飾子	public	制限はありません。どこからでもアクセス可能。
	protected	同一パッケージ内、またはそのサブクラスからのみアクセス可能。
	private	同一クラス内でのみアクセス可能。
修飾子	abstract	抽象メソッドや抽象メソッドをもつクラスに指定する。インターフェースは元々抽象メソッドしか定義できない為、省略されることが多い。
	final	クラスにつける場合：クラスを継承することを禁止 変数につける場合：定数の定義
	strictfp	「float」や「double」の浮動小数点を用いた演算を厳密に行いたいクラスに指定する。
	static	クラス固有のものとして、同一クラスからなる全インスタンスに共有させる場合に指定する。 静的メソッドや静的変数を表す修飾子です。

	transient	シリアライズ（バイトストリーム変換）の時に値を保存したくない（一時的に使用する値・初期化したい値）フィールドに指定する。
	volatile	コンパイラによる最適化をさせたくないフィールドに指定します。（マルチスレッド等による参照値の相違をなくします。）
	native	プラットフォーム依存のコードで定義します。（型、メソッド名、引数の型といったインターフェースにみを定義し、中身はC言語などの他の言語を用いて実装します。）
	synchronized	マルチスレッド環境において、同期処理（インスタンス単位で排他制御）させる場合に指定する。