Javaプログラミング実習

15. クラスの利用

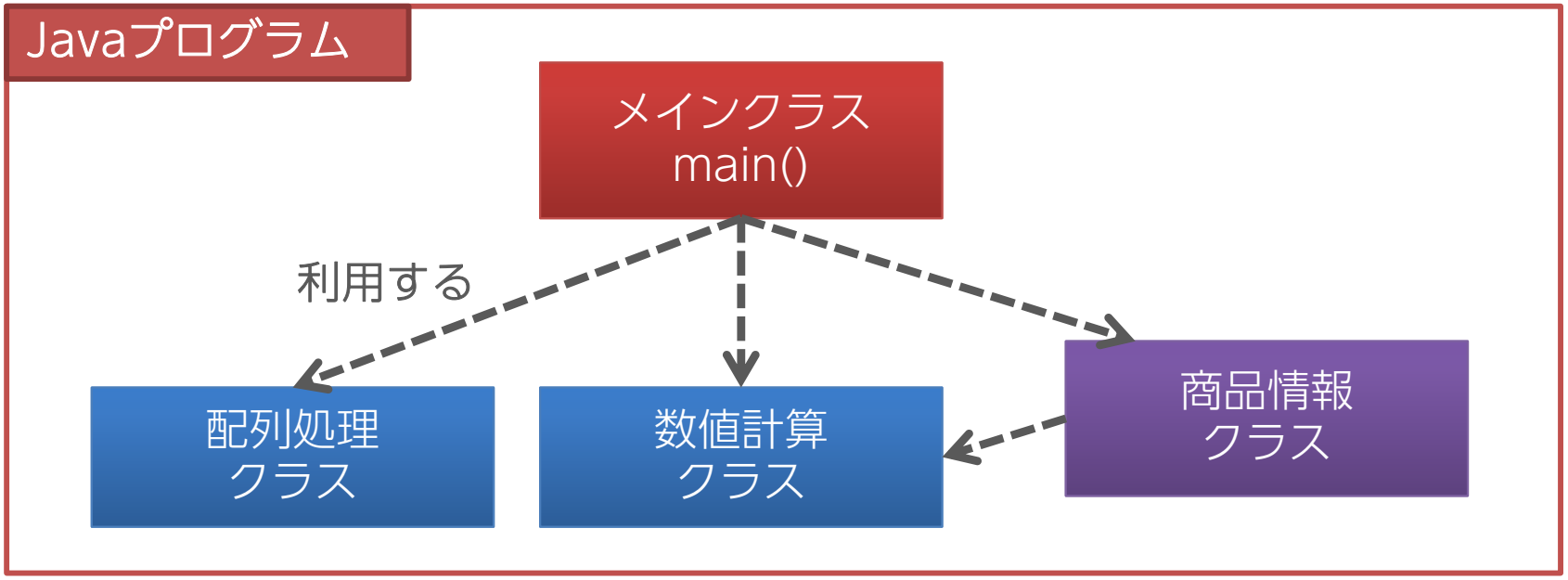
株式会社ジードライブ

今回学ぶこと

- ソースファイルの分割
- クラスの利用
- パッケージ

ソースファイルの分割

- Javaプログラムは複数のクラスで構成することができる
  - 幾つかの変数やメソッドをまとめて、別ファイル(別クラス)として管理することができる



あるクラスから、他のクラスで定義されているメソッドを呼び出したり、
他のクラスで定義されている変数や定数を参照することができる

クラスとは

- ある特定の事柄に関する情報(変数／定数)と機能(メソッド)をもつプログラムのモジュール(部品)
  - クラスは別のクラスから利用することができる
  - 既存のクラスを利用するだけでなく、自身でクラスを作成することも可能

数値計算クラス

```
static final double PI

static int max(int a, int b) { … }
static double sqrt(double a) { … }
                              :
```

内包する変数／定数

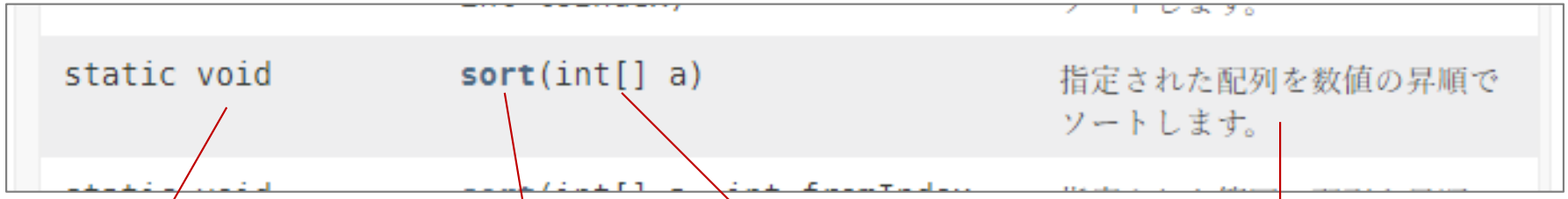
内包するメソッド

※ モジュール…建築材料・家具などの規格化された組み立てユニット。
また、装置・機械・システムを構成する、機能的にまとまった部分。(引用：Wikipedia)

Java API ドキュメントとは

- 標準的に含まれるパッケージ/クラス/メソッド等の説明書 (Oracleによる公式の資料)
  - https://docs.oracle.com/javase/jp/21/docs/api/
- クラスごとにメソッドの説明を見ることができる
  - メソッドの概要、戻り値、引数など

例: Arraysクラスの説明
https://docs.oracle.com/javase/jp/21/docs/api/java.base/java/util/Arrays.html



他のクラスの利用方法①

- 他のクラスで定義されているメソッドや変数等を利用する場合、クラス名とドット記号でつなげて記述する

書式

```
クラス名.メソッド名(引数, …)
クラス名.変数名
クラス名.定数名
```

他のクラスの利用例

- 例：数値計算クラス Math の利用
  - max()メソッド：2つの値のうち大きな値を返す

```
public class MyApp {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = Math.max(a, b);
        System.out.println("c = " + c);
    }
}
```

コンソール表示

```
c = 20
```

他のクラスの利用例

- メソッドと同様、他のクラスで定義されている変数や定数も利用することができる
  - 例：Mathクラスの定数PIを利用

```
public class MyApp {
    public static void main(String[] args) {
        double radius = 2.5;
        double length = 2.0 * Math.PI * radius;
        System.out.println(length);
    }
}
```

コンソール表示

```
15.707963267948966
```

他のクラスの利用方法②

- 他クラスのメソッドを利用するにあたっては、以下のようなパターンも多く存在する

書式

```
クラス型　変数　= new クラス(引数, …);
変数.メソッド名(引数, …);
```

例

```
Scanner sc = new Scanner(System.in);
String tx = sc.nextLine();
```

このパターンについては、後の章で学習する

練習

- 練習15-1

アクセス修飾子

- クラスの変数やメソッドについている **public** や **private** などのキーワードを**アクセス修飾子**と呼ぶ
- アクセス修飾子は、変数やメソッドが他のクラスから利用できるかどうかを示している

アクセス修飾子の例

```
public static void main(String[] args) { … }

private static double calcBMI(double h, double w) { … }
```

アクセス修飾子

- **public** が付いている変数やメソッドは他のクラスから利用できる
- **private** が付いている変数やメソッドは、自クラス内では利用できるが他のクラスからは利用できない

アクセス修飾子による利用可能範囲

public	他のクラスから利用可能
private	自クラス内でのみ利用可能

クラス変数とクラスメソッド

- **static** キーワードの付いた変数／メソッドをそれぞれ、クラス変数／クラスメソッドと呼ぶ
  - **static** キーワードの付かないものは、インスタンス変数／インスタンスメソッドと呼ばれる
- クラス変数／メソッドは、インスタンスを生成しなくても利用することができる

staticやインスタンスについては、
オブジェクト指向の単元で学習します

オーバーロード

- Javaでは1つのクラス内で同じ名前のメソッドを複数定義することができる
  - この仕組みをオーバーロードという
- あるメソッドについて、引数の型や引数の数にいくつかのバリエーションをもたせたい場面で利用する
  - オーバーロードする場合には、**引数の型**か**引数の数**を変える必要がある
  - 引数の型や数が同じで、戻り値だけが異なるようなオーバーロードはできない

オーバーロードの例

- 例：税率が決まっているものと、任意の税率を設定できるものの２パターンのメソッド

```
public class Tax {
  private static int priceWithTax(int price) {
    int result = (int) (price * 1.1);
    return result;
  }
  private static int priceWithTax(int price, double taxRate) {
    int result = (int) (price * (1 + taxRate / 100));
    return result;
  }
  public static void main(String[] args) {
    double a = priceWithTax(1000);       // 税率10% ⇒ 1100
    double b = priceWithTax(1000, 15);   // 税率15% ⇒ 1150
  }
}
```

同名のメソッドだが、
引数の数が異なる

練習

- 練習15-2

パッケージ

- パッケージとは、複数のクラスをひとまとめにしたもの
- Javaでは、標準で利用できる多くのパッケージが用意されている

標準パッケージの例

パッケージ	内容
`java.lang`	Java言語の基本的なクラスを集めたパッケージ
`java.util`	配列処理、データ集合処理、日時処理など汎用的で利便性の高い機能を有するクラスの集まり
`java.io`	データ入出力を担当するクラスの集まり
`java.net`	ネットワーク関連を担当するクラスの集まり

Javaの標準パッケージは `java` で始まる

パッケージ

- 自分でクラスを作る場合も、パッケージを指定する(宣言する)
  - クラスは必ずどこかのパッケージに属する必要がある
  - 宣言されたパッケージと、クラスの保存先フォルダは一致する必要がある
  - パッケージの宣言は、コードの冒頭に記述する

書式

```
package パッケージ名;
```

例

```
package practice01;

public class MyApp {
    …
```

パッケージのルール

- パッケージはドット記号(.)で区切られた階層命名パターンを使用する
  - パッケージ名とフォルダの階層関係は一致する
  - 例：`mypack.utility.math`というパッケージに属するクラスは、`mypack/utility/math`フォルダ内に保存する
- グローバルなパッケージ（クラスを外部に公開して使用してもらう場合）はインターネットドメインに基づいて命名する
  - 例： **`example.com`** というドメインを持つ組織で作成するパッケージの場合、**`com.example.myapp`** といったパッケージ名となる
  - パッケージ名として使用不可の文字（ハイフンなど）がドメイン名に含まれる場合は、アンダースコアで代用する

クラスのimport

- 他のクラスを利用する場合、パッケージ名を明示する必要がある
  - パッケージ名とクラス名をドット記号で繋げて記述する
  - パッケージ名を含むクラス名をFQCN(Fully Qualified Class Name)という

例： java.utilパッケージに属するArraysクラスのtoStringメソッドの利用

```
public class MyApp {
  public static void main(String[] args) {
    String[] fruits = {"りんご", "バナナ", "ぶどう"};
    System.out.println(java.util.Arrays.toString(fruits));
  }
}
```

クラスのimport

- import宣言を記述することで、FQCNによるクラス名の記述が不要になる
  - 同じパッケージに所属するクラス、及びjava.lang パッケージに含まれるクラスについては、import文を省略できる

書式

```
import パッケージ名.クラス名;
```

```
import java.util.Arrays;

public class MyApp {
  public static void main(String[] args) {
    String[] fruits = {"りんご", "バナナ", "ぶどう"};
    System.out.println(Arrays.toString(fruits));
  }
}
```

import宣言の記述はEclipseで行う
⇒ Ctrl + Shift + O (インポートの編成)

ワイルドカードによるimport

- import時にアスタリスク記号を記述することで、特定のクラスではなく、全てのクラスを指定することができる
  - ただし、この方法は推奨されない

```
import java.util.*;

public class MyApp {
  public static void main(String[] args) {
    String[] fruits = {"りんご", "バナナ", "ぶどう"};
    System.out.println(Arrays.toString(fruits));
  }
}
```

staticインポート

- import文でクラス変数やクラスメソッドまで指定すると、クラス名を省略してクラス変数やメソッドが利用できる
  - staticインポートと呼ばれるが、あまり使用されない

書式

```
import static パッケージ名.クラス名.メソッド(あるいは変数・定数)名;
```

「パッケージ名.クラス名.*」の様にワイルドカード を使用することもできる

例

```
import static java.util.Arrays.sort;

public class MyApp {
  public static void main(String[] args) {
    int[] a = {5, 3, 11};
    sort(a);  // java.util.Arrays.sort() を使用
...
```

練習

- 練習15-3