コンストラクタとカプセル化（この単元）— Lesson 9・約2時間

この単元の前に：オブジェクト指向の復習

前回は「クラスとインスタンス」「フィールドとメソッド」を自販機で学びました。まず、その部分を短くおさらいします。

復習：前回までに学んだこと

- **クラスとインスタンス** — クラスは設計図、インスタンスは `new クラス名()` で作った実物です。
- **フィールドとメソッド** — フィールドは「状態（データ）」、メソッドは「動き（処理）」です。メソッドの中でフィールドを使います。
- **定義する側と呼び出す側** — クラスを書くファイル（定義側）と、`main` で `new` してメソッドを呼ぶファイル（呼び出し側）で役割を分けます。

この単元で学ぶことのイメージ

この単元では「コンストラクタ」「カプセル化」「this」を詳しく学びます。ここでは、最初にイメージだけ伝えておきます。

- **コンストラクタ** — インスタンスが作られるときに一度だけ実行される特別なメソッドです。クラス名と同じ名前で、戻り値の型を書きません。初期値をセットするのに使います。
- **カプセル化** — フィールドを `private` にして外から直接触れないようにし、`public` なメソッド（getter・setter）だけから操作する考え方です。
- **this** — メソッドやコンストラクタの中で「このインスタンス自身」を指すキーワードです。フィールド名と引数名が同じときに `this.フィールド名` で区別します。

この単元では、自販機に「コンストラクタ」と「カプセル化」を追加して、より安全で分かりやすいプログラムにしていきます。

【1】具体的な学習目標

この単元を終えたとき、以下のことができればゴールです。

【必須】

- コンストラクタの役割を説明できる。
- コンストラクタでインスタンスの初期値（商品名・在庫など）をセットできる。
- コンストラクタのオーバーロード（引数の違う複数のコンストラクタ）を書ける。
- カプセル化の目的（中身を隠して安全にすること）を説明できる。
- `private` フィールドと getter・setter の役割を説明できる。

【オプション】

- `this` を使ってフィールドと引数を区別できる。

【2】導入：自販機を「設置する瞬間」に決める

> **カバ先生：**「前回は、自販機を作ったあとで `name = "水"` のようにフィールドに直接代入していたね。でも本当の自販機は、設置する瞬間に『これはコーラ専用機だよ』『在庫は10本から』と決まっていることが多い。その『設置する瞬間に実行される処理』がコンストラクタなんだ。」

> **ナナコ：**「`new VendMachine()` で作るときに、いっぺんに名前や在庫を渡せるようにする、ってことですね。」

> **ユウタ：**「それなら、呼び出す側で `name` を書き忘れたり、変な値を入れちゃう心配が減りそう！」

> **カバ先生：**「その通り。あとで『中身をむき出しにしない』カプセル化もやって、さらに安全にしていこう。」

【3】コンストラクタで自販機を初期設定する

コンストラクタとは

コンストラクタは、`new クラス名()` でインスタンスが作られるときに**一度だけ**自動で実行される特別なメソッドです。

- 名前は**クラス名と完全に同じ**にします。
- 戻り値の型は書きません（`void` も不要）。
- 引数を受け取れるので、「コーラ専用機」「在庫10」のように初期値を渡せます。

サンプル：VendMachine にコンストラクタを追加

```
public class VendMachine {

    String name;  // あとでコンストラクタでセットする
    int stock;

    // コンストラクタ：インスタンスが作られるときに1回だけ実行される
    public VendMachine(String name, int num) {
        this.name = name;
        stock = num;
    }

    public void buy() {
        if (stock > 0) {
            System.out.println(name + "を購入しました");
            stock--;
            System.out.println("残りの在庫は" + stock + "個です");
        } else {
            System.out.println("売り切れです");
        }
    }
}
```

呼び出す側（Main）

```
public class Main {

    public static void main(String[] args) {
        // 設置する瞬間に「コーラ」「在庫10」を渡す
        VendMachine vm = new VendMachine("コーラ", 10);
        vm.buy();
        vm.buy();

        // 別の自販機は「お茶」「在庫5」
        VendMachine vm2 = new VendMachine("お茶", 5);
        vm2.buy();
    }
}
```

> **ナナコ：**「`new VendMachine("コーラ", 10)` で、作った瞬間に名前と在庫が決まるのね。見やすい！」

【4】コンストラクタのオーバーロード

同じクラスに、**引数の種類や個数が違う**コンストラクタを複数用意することができます。これをコンストラクタのオーバーロードといいます。

例：引数なしのときは「水」「在庫10」にする。

```
public VendMachine() {
    name = "水";
    stock = 10;
}

public VendMachine(String name, int num) {
    this.name = name;
    stock = num;
}
```

呼び出し例：

```
VendMachine vm = new VendMachine();           // 水, 10
VendMachine vm2 = new VendMachine("コーラ", 10); // コーラ, 10
```

【5】練習問題（コンストラクタ）

【問題1】コンストラクタで初期化する

`VendMachine` クラスに、飲み物の名前（`String`）と初期在庫数（`int`）を受け取るコンストラクタを1つ追加してください。フィールド `name` と `stock` に、受け取った値を代入するようにしてください。

※ヒント・解答は末尾の「ヒント・解答一覧」を参照。

【問題2】コンストラクタのオーバーロード

引数を受け取らないコンストラクタを1つ追加してください。そのとき `name` を「水」、`stock` を 10 に初期化するようにしてください。

※ヒント・解答は末尾の「ヒント・解答一覧」を参照。

【6】カプセル化：自販機のカバーをかける

> **カバ先生：**「今は `vm.name = "ドロ水"` のように、外から好き勝手にフィールドを書き換えられてしまう。自販機でいうと、中の商品を誰かが勝手にいじるようなものだね。そこで、フィールドを『中だけの秘密』にして、外にはボタン（メソッド）からしか触らせないようにする。これがカプセル化だよ。」

private と getter・setter

- `private` — そのフィールドは**クラスの内側からだけ**アクセスできます。外からは見えません。
- **getter** — フィールドの値を**読む**ための `public` メソッド（例：`getName()`）。
- **setter** — フィールドの値を**書き換える**ための `public` メソッド（例：`setStock(int n)`）。必要に応じて「0未満はセットしない」などのルールを入れられます。

サンプル：フィールドを private にして getter を用意

```
public class VendMachine {

    private String name;  // 外から直接触れない
    private int stock;

    public VendMachine(String name, int num) {
        this.name = name;
        stock = num;
    }

    // getter：飲み物の名前を返す
    public String getName() { return name; }
    // getter：在庫数を返す
    public int getStock() { return stock; }

    public void buy() {
        if (stock > 0) {
            System.out.println(name + "を購入しました");
            stock--;
            System.out.println("残りの在庫は" + stock + "個です");
        } else {
            System.out.println("売り切れです");
        }
    }
}
```

呼び出す側では、`vm.name` は使えません。代わりに `vm.getName()` で値を取得します。

【7】this で「このインスタンス」を指す

コンストラクタの引数名をフィールドと同じにすると、どちらがフィールドか分かりにくくなります。そんなとき `this.フィールド名` で「このインスタンスのフィールド」をはっきり指せます。

```
public VendMachine(String name, int stock) {
    this.name = name;  // 左：このインスタンスのフィールド、右：引数
    this.stock = stock;
}
```

> **ユウタ：**「`this` をつけると、フィールドだと分かるんだね。」

【8】練習問題（カプセル化）

【問題3】private と getter を追加する

`VendMachine` のフィールド `name` と `stock` を `private` に変更し、それぞれの値を返す getter `getName()` と `getStock()` を追加してください。

※ヒント・解答は末尾の「ヒント・解答一覧」を参照。

【問題4】setter で在庫を安全に更新する

在庫数を設定する setter `setStock(int n)` を追加してください。ただし、`n` が 0 未満のときは在庫を変更しないようにしてください。

※ヒント・解答は末尾の「ヒント・解答一覧」を参照。

【9】練習問題（自販機の拡張）

ここでは、自販機クラスにフィールドを追加する形で、コンストラクタとカプセル化の考え方をくり返し練習します。題材は自販機のまま統一します。

価格フィールドの追加

【問題5】VendMachine に「1本の価格」を追加する

`VendMachine` に、private フィールド `int price`、コンストラクタのオーバーロード `VendMachine(String name, int stock, int price)`、getter `getPrice()` を追加してください。

※ヒント・解答は末尾の「ヒント・解答一覧」を参照。

機械番号フィールドの追加

【問題6】VendMachine に「機械番号」を追加する

`VendMachine` に、private フィールド `int machineId`、コンストラクタのオーバーロード `VendMachine(String name, int stock, int machineId)`、getter `getMachineId()` を追加してください。

※ヒント・解答は末尾の「ヒント・解答一覧」を参照。

【10】まとめのおさらい

- コンストラクタは、`new` でインスタンスが作られるときに1回だけ実行される。クラス名と同じ名前で、戻り値の型を書かない。
- コンストラクタに引数を渡すと、設置する瞬間に「商品名」「在庫」などを初期値としてセットできる。
- コンストラクタのオーバーロードで、引数なし・引数ありなど、複数の初期化パターンを用意できる。
- カプセル化：フィールドを `private` にして中身を隠し、getter・setter でだけ操作する。安全で分かりやすいプログラムになる。
- `this` は「このインスタンス自身」を指し、フィールドと引数を区別するときに使う。

> **ナナコ：**「コンストラクタで初期設定、カプセル化で安全に。自販機のイメージで覚えられそう。」

> **カバ先生：**「次は、自販機の『型』を広げる話、継承とオーバーライドに進んでいこう！」

【ヒント・解答一覧】

問題と対応するヒント・解答例です。行き来して参照してください。

【問題1】コンストラクタで初期化する

ヒント

コンストラクタの名前はクラス名 `VendMachine` と完全に同じにします。引数は `(String name, int num)` のように2つ受け取り、中で `this.name = name;` と `stock = num;` のように代入します。

解答例

```
public VendMachine(String name, int num) {
    this.name = name;
    stock = num;
}
```

【問題2】コンストラクタのオーバーロード

ヒント

引数なしのコンストラクタは `public VendMachine() { ... }` と書きます。中で `name = "水";` と `stock = 10;` を代入します。

解答例

```
public VendMachine() {
    name = "水";
    stock = 10;
}
```

【問題3】private と getter を追加する

ヒント

フィールドの前に `private` を付けます。getter は `return フィールド名;` で値を返します。

解答例

```
private String name;
private int stock;
public String getName() { return name; }
public int getStock() { return stock; }
```

【問題4】setter で在庫を安全に更新する

ヒント

`public void setStock(int n) { ... }` とし、中で `if (n >= 0) { stock = n; }` のようにします。

解答例

```
public void setStock(int n) {
    if (n >= 0) { stock = n; }
}
```

【問題5】VendMachine に「1本の価格」を追加する

ヒント

コンストラクタでは `this.name = name;`、`this.stock = stock;`、`this.price = price;` のように代入します。getter は `return price;` で値を返します。

解答例

```
private int price;

public VendMachine(String name, int stock, int price) {
    this.name = name;
    this.stock = stock;
    this.price = price;
}

public int getPrice() {
    return price;
}
```

【問題6】VendMachine に「機械番号」を追加する

ヒント

コンストラクタでは `this.name = name;`、`this.stock = stock;`、`this.machineId = machineId;` のように代入します。getter は `return machineId;` で値を返します。

解答例

```
private int machineId;

public VendMachine(String name, int stock, int machineId) {
    this.name = name;
    this.stock = stock;
    this.machineId = machineId;
}

public int getMachineId() {
    return machineId;
}
```