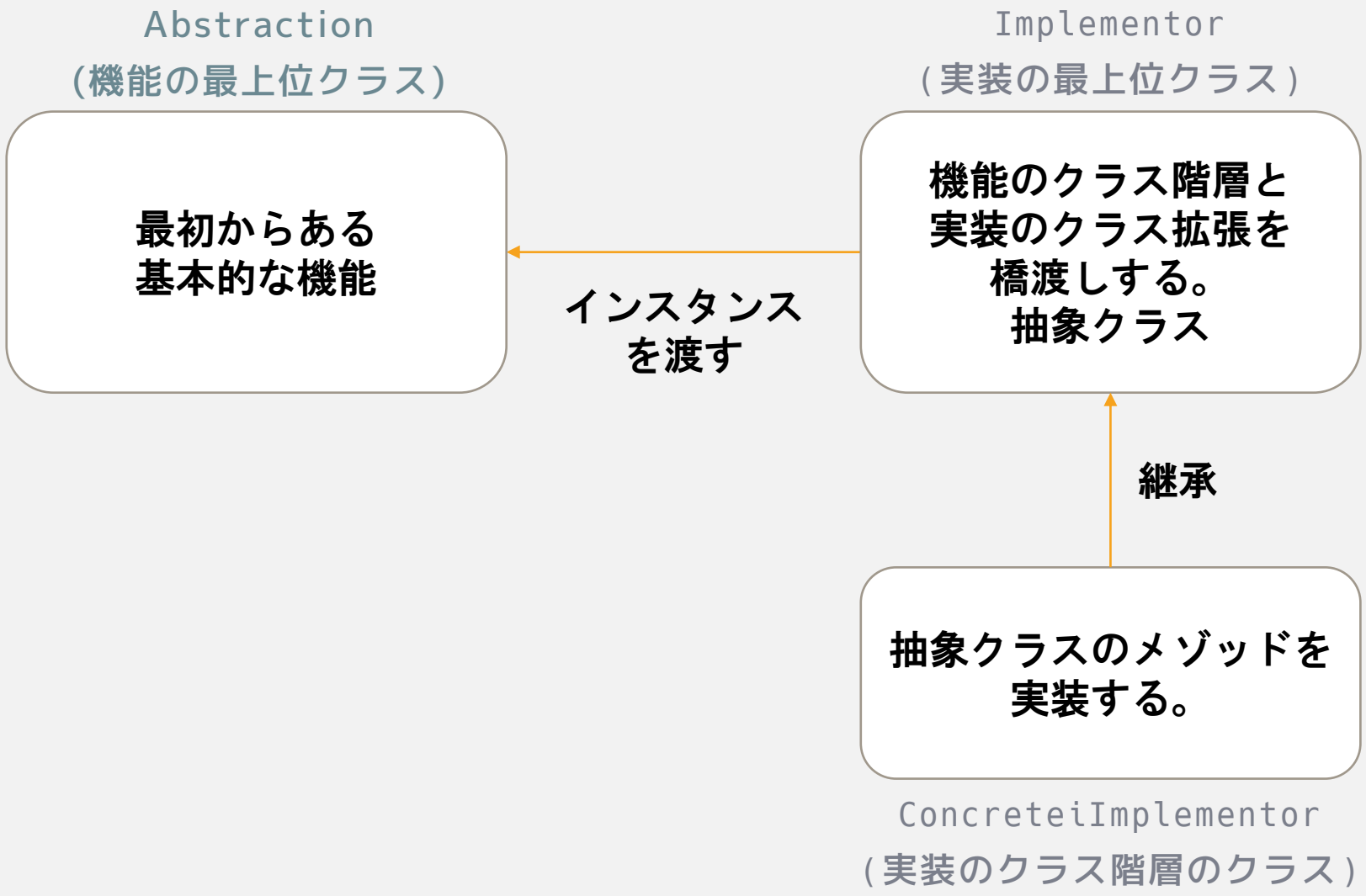
デザインパターン「9-Bridge」

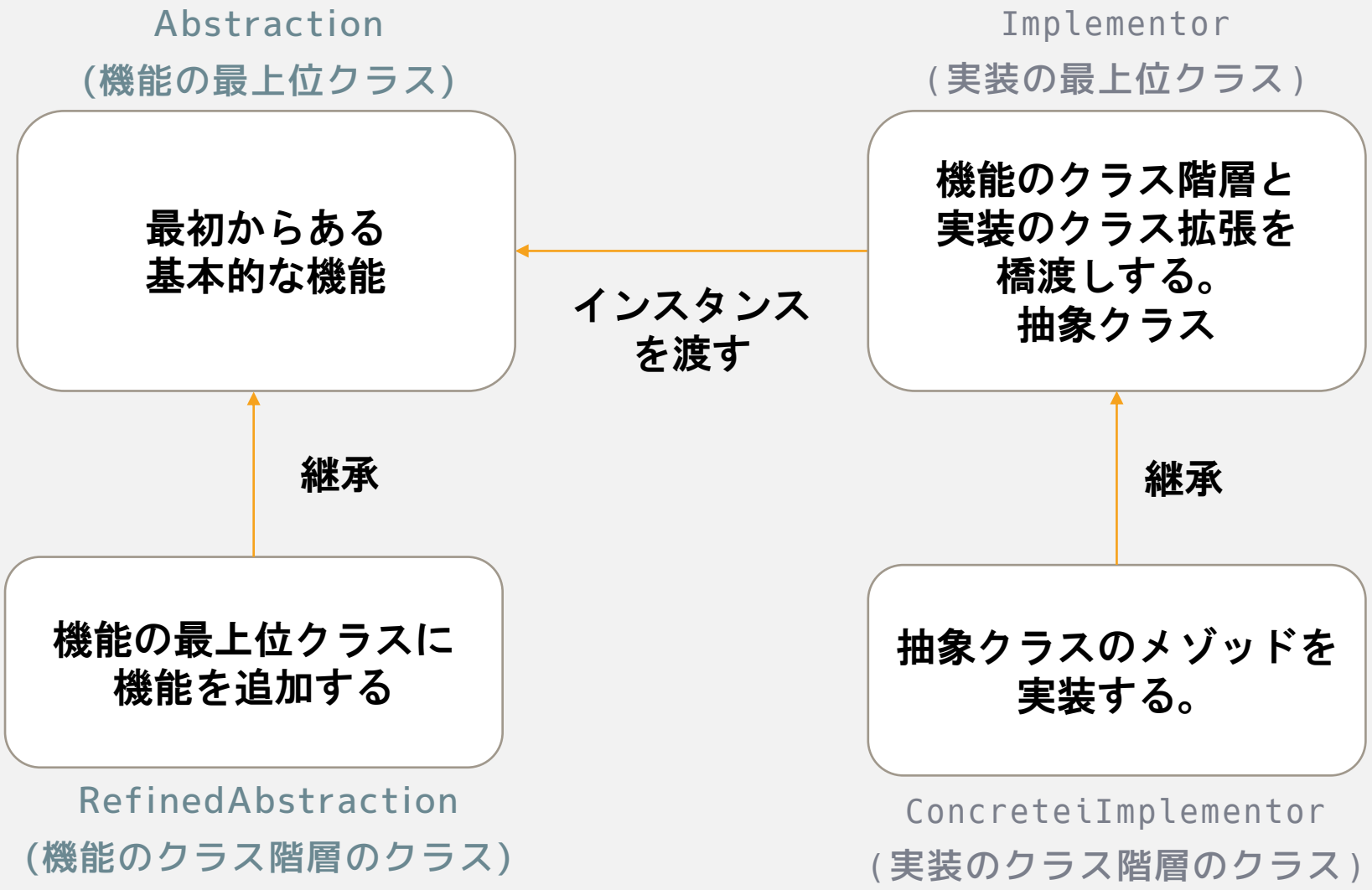
Seeeeee:D　夏休み勉強会

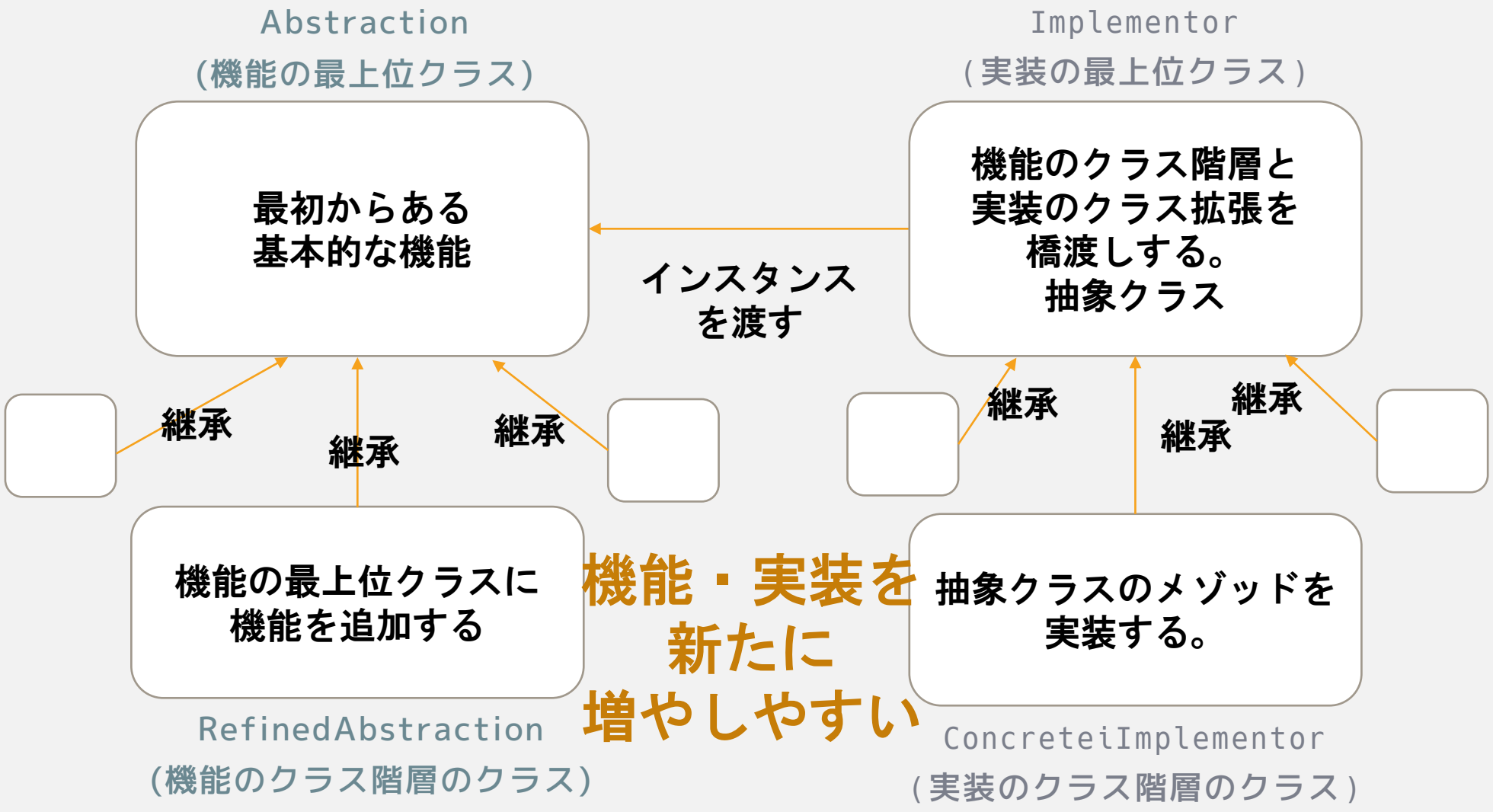
いつ使うの??

- 見通しよく、クラス拡張ができる。
- 実装のクラス階層と機能のクラス階層とを分けている。
- 機能に影響を与えずに実装を拡張できる。
- 実装に影響を与えずに機能を拡張できる。

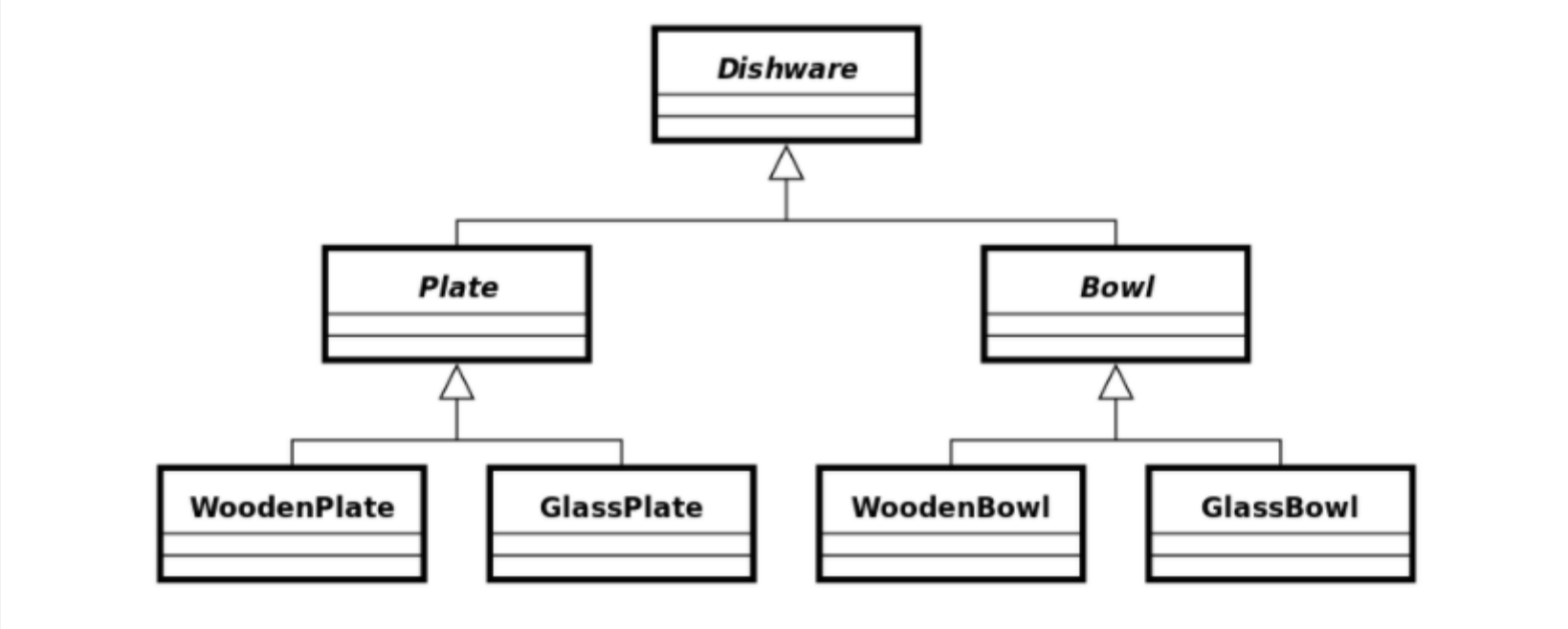
用語説明


Abstraction
(機能の最上位クラス)
最初からある
基本的な機能
インスタンス
を渡す
Implementor
（実装の最上位クラス）
機能のクラス階層と
実装のクラス拡張を
橋渡しする。
抽象クラス
継承
抽象クラスのメゾッドを
実装する。
ConcreteiImplementor
（実装のクラス階層のクラス）


Abstraction
(機能の最上位クラス)
最初からある
基本的な機能
Implementor
(実装の最上位クラス)
機能のクラス階層と
実装のクラス拡張を
橋渡しする。
抽象クラス
インスタンス
を渡す
継承
継承
機能の最上位クラスに
機能を追加する
抽象クラスのメゾッドを
実装する。
RefinedAbstraction
(機能のクラス階層のクラス)
ConcreteiImplementor
(実装のクラス階層のクラス)

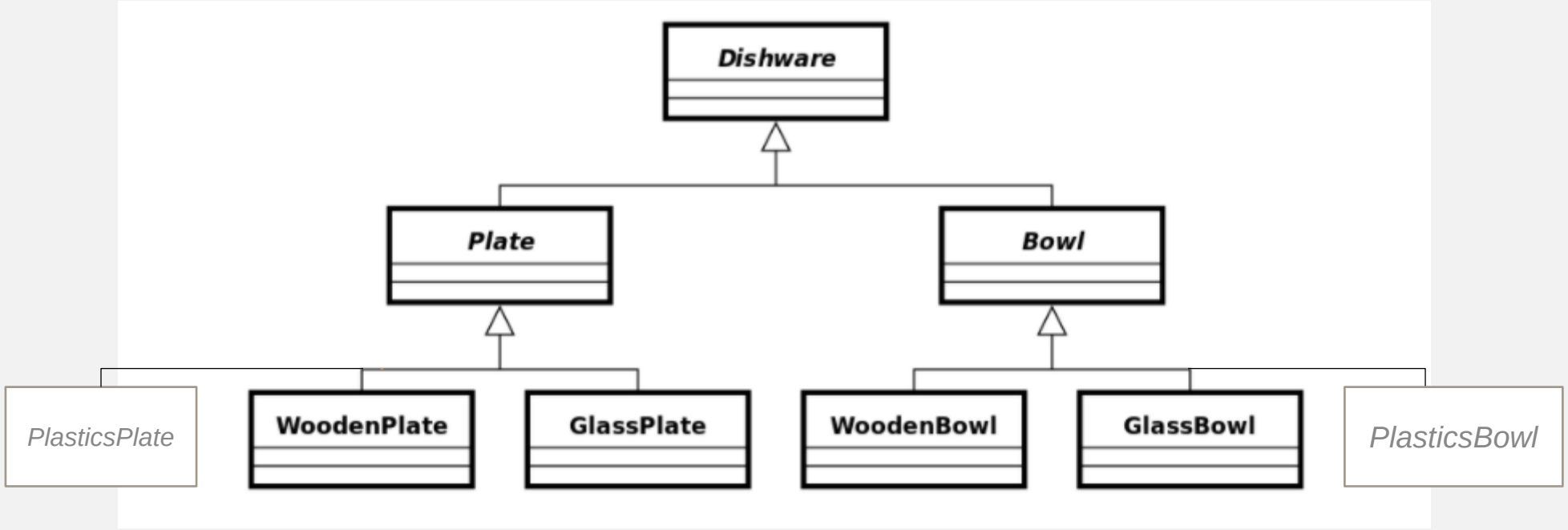

Abstraction
(機能の最上位クラス)
Implementor
(実装の最上位クラス)
最初からある
基本的な機能
インスタンス
を渡す
機能のクラス階層と
実装のクラス拡張を
橋渡しする。
抽象クラス
継承
継承
継承
継承
継承
機能の最上位クラスに
機能を追加する
機能・実装を
新たに
増やしやすい
抽象クラスのメゾッドを
実装する。
RefinedAbstraction
(機能のクラス階層のクラス)
ConcreteiImplementor
(実装のクラス階層のクラス)

BRRIDGEを使わない場合

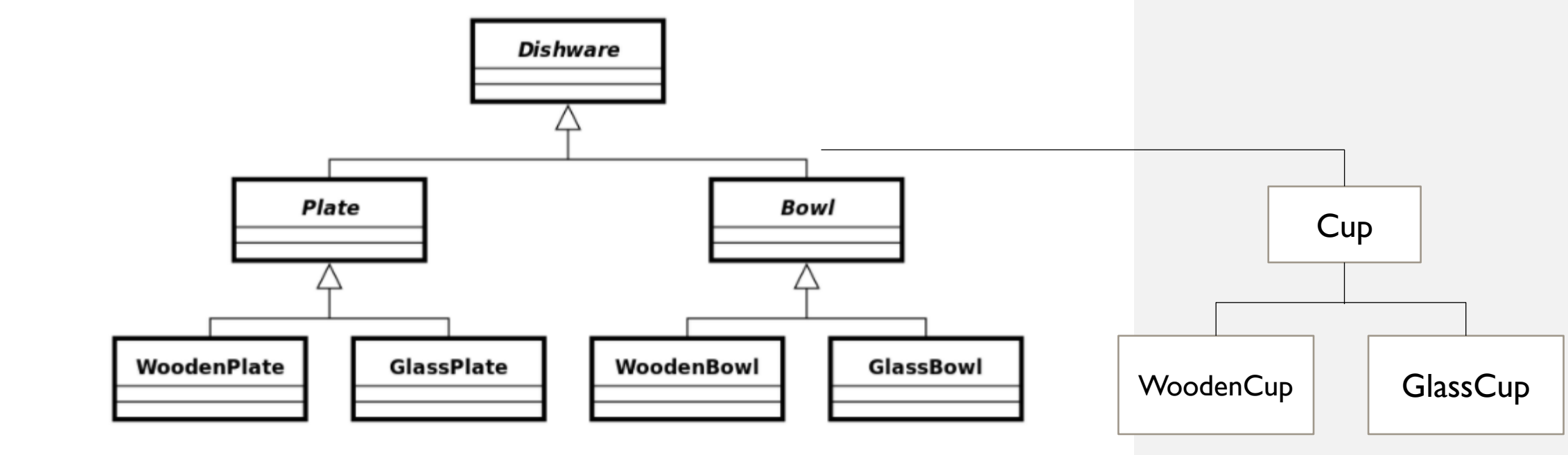


出典：https://ja.wikipedia.org/wiki/Bridge_%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3

BRRIDGEを使わない場合

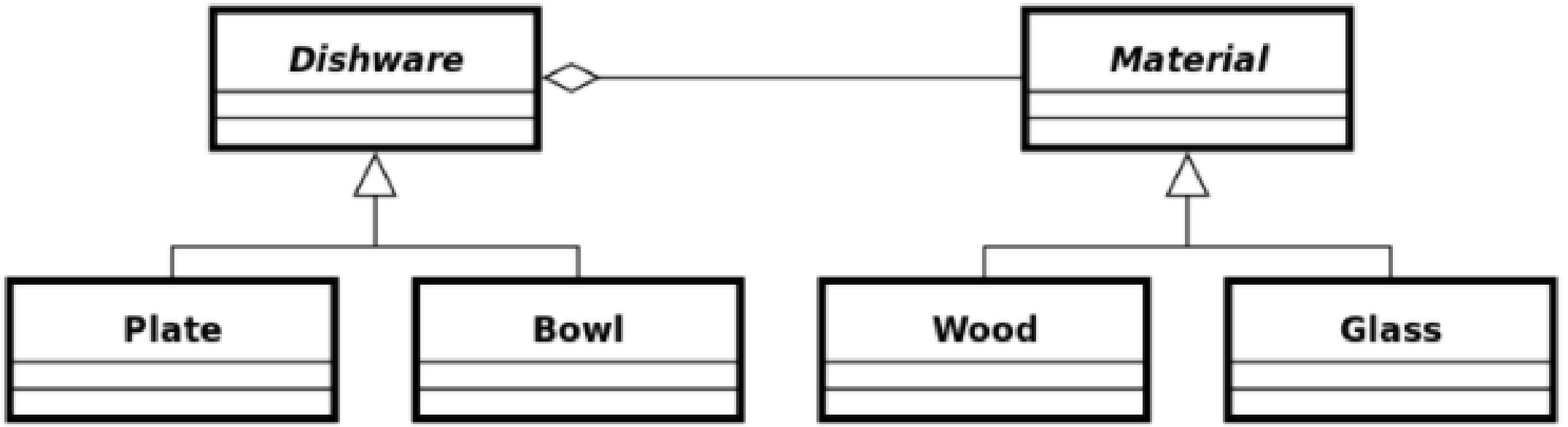


BRRIDGEを使わない場合



出典：https://ja.wikipedia.org/wiki/Bridge_%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3

BRRIDGEを使えば



種類(機能)も材質(実装)も追加しやすい

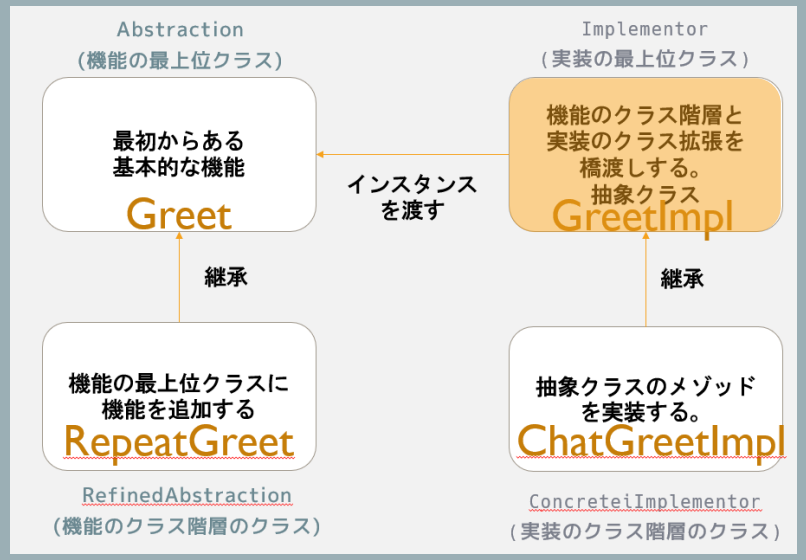
出典：https://ja.wikipedia.org/wiki/Bridge_%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3

具体例

Implementor

実装の最上位クラス。

実装と機能の橋渡し

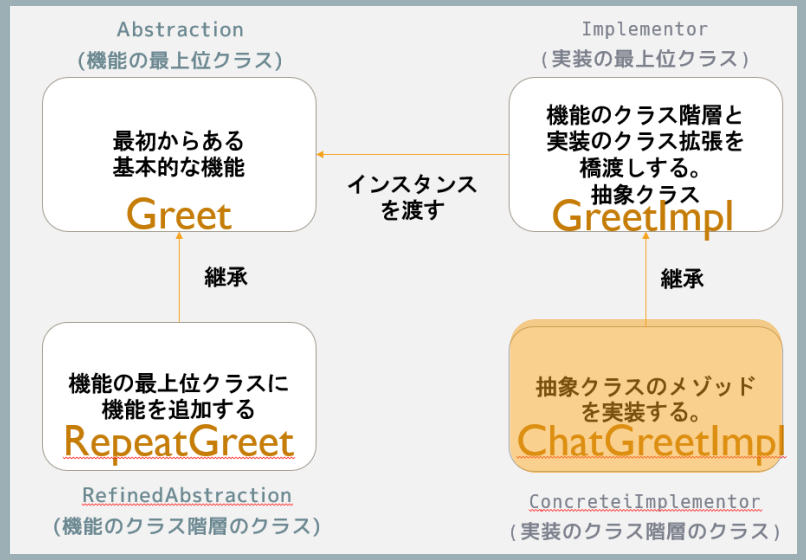



```ts
export default abstract class GreetImpl {
    // protected : 継承クラス内でのみ参照可能
    protected message: string;

    constructor(message: string) {
        this.message = message;
    }
    // abstract : 子クラスに実装を任せる。
    abstract start(): void;
    abstract print(): void;
    abstract end(): void;
}
```

ConcreteImplementor

実装のクラス階層のクラス。

具体的な実装

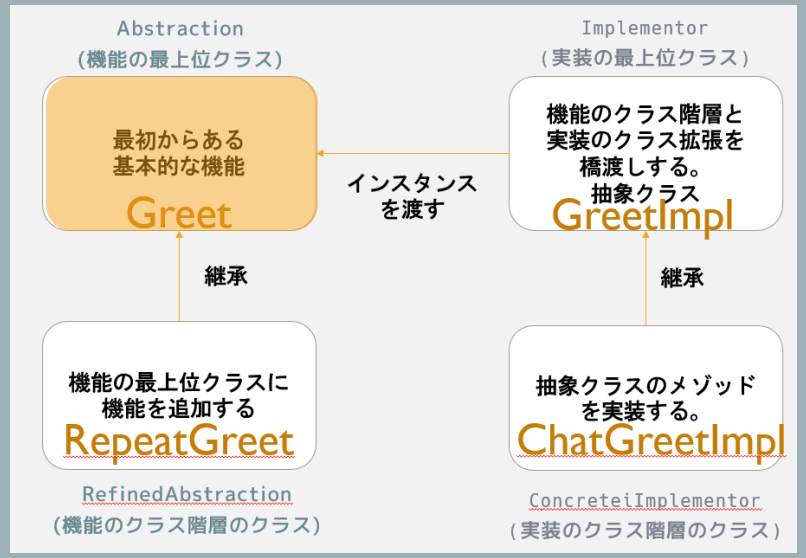



```
import GreetImpl from "./GreetImpl";
export default class ChatGreetImpl extends GreetImpl {
    constructor(message: string) {
        super(message);  // 親(GreetImpl)のconstructorを呼び出す。
    }
    //以下具体的な実装内容
    start(): void {
        this.printLine();
    }
    print(): void {
        console.log(this.message);
    }
    end(): void {
        this.printLine();
    }
    private printLine(): void {
        const messageCount: number = this.message.length;
        let line: string = '';
        for (let i: number = 0; i < messageCount; i++) {
            line += '-';
        }
        console.log(line);
    }
}
```

Abstraction

機能の最上位クラス。

最初からある基本的な機能

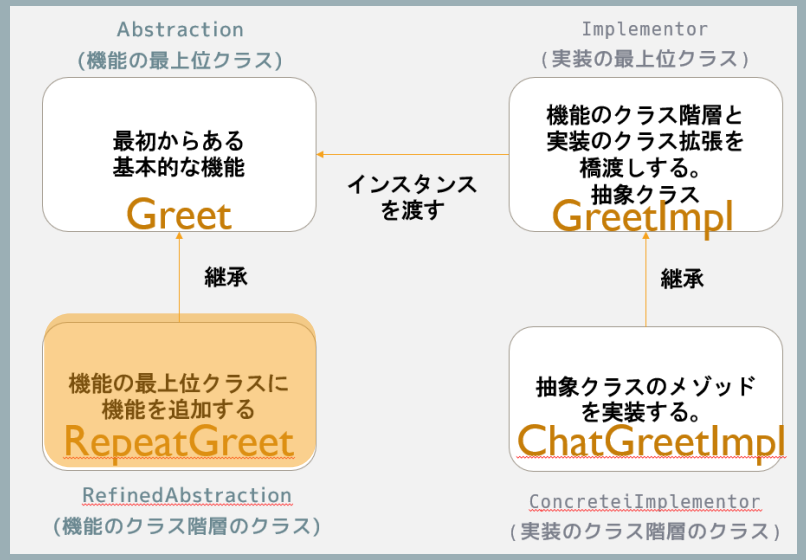



```
import GreetImpl from "./GreetImpl";

export default class Greet {
    // クラス変数を定義
    private impl: GreetImpl;

    //橋(Impl)を受け取る
    constructor(tmp_impl: GreetImpl) {
        this.impl = tmp_impl;
    }

    start(): void {
        this.impl.start();
    }
    print(): void {
        this.impl.print();
    }
    end(): void {
        this.impl.end();
    }
    display(): void {
        this.start();
        this.print();
        this.end();
    }
}
```

RefinedAbstraction

機能のクラス階層のクラス。

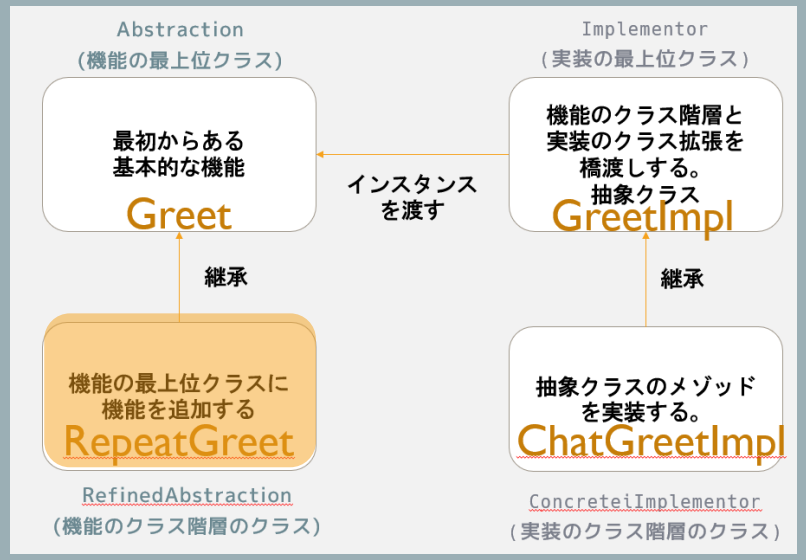
機能を追加する。




```
import Greet from "./Greet";
import GreetImpl from "./GreetImpl";

export default class RepeatGreet extends Greet {
    constructor(impl: GreetImpl) {
        super(impl);
    }

    // Greet クラスに機能を追加。
    repeatDisplay(): void {
        this.start();
        this.print();
        this.print();
        this.end();
    }
}
```

Main

実際の呼び出し



Main.ts

```ts
import Greet from "./modules/Greet";
import ChatGreetImpl from "./modules/ChatGreetImpl";
import RepeatGreet from "./modules/RepeatGreet";
// 機能クラス(実装クラス)
const greet1: Greet = new Greet(new ChatGreetImpl('Good Morning'));
const greet2: Greet = new RepeatGreet(new ChatGreetImpl('Hello'));
const greet3: RepeatGreet = new RepeatGreet(new ChatGreetImpl('Good Evening'));

greet1.display();
greet2.display();
greet3.display();
greet3.repeatDisplay();
```

output

```
------------
Good Morning
------------
-----
Hello
-----
------------
Good Evening
------------
------------
Good Evening
Good Evening
------------
```

まとめ

- スマートに実装も機能も拡張できる。
- 使いこなすの難しそうだけど実用的に感じた。
- TypeScriptが便利すぎる。コード補完たくさんしてくれる。