

# 第22章 インターフェイス

## 目次

- インターフェイス
- 抽象クラスとインターフェイスの違い
- インターフェイスの拡張

## インターフェイスとは

オブジェクトを利用する側に公開すべき操作(メソッド)を  
まとめたクラスのこと。

## インターフェイスとは

異なる処理内容ではあるが、目的は同じという機能の場合、オブジェクトに対して共通のインターフェイスを定義することで操作方法が統一され利便性が上がる。

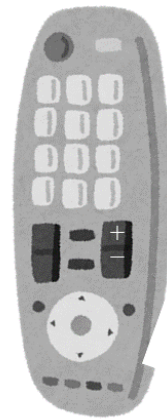
多態性(ポリモーフィズム):

共通部分を定義しながらも、それぞれのクラスごとに独自の処理を定義するプログラミング手法

## (例) テレビのリモコン: インターフェイスあり

どのメーカー、どのタイプのリモコンを使ったとしても、  
操作の方法に大きな違いはない。

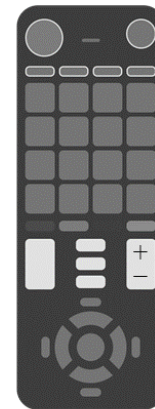
- 電源ボタンを押す: 電源が入る
- +ボタンを押す: 音量が上がる



A社のリモコン



B社のリモコン



C社のリモコン

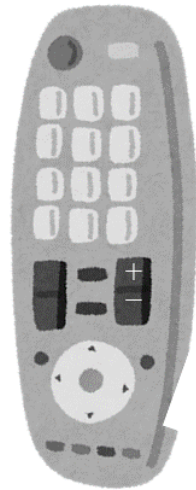


## (例) テレビのリモコン: インターフェイスなし

音量を上げたい場合...

A社のリモコンは●ボタン

B社のリモコンはdボタン など



A社のリモコン



B社のリモコン



C社のリモコン



## インターフェイスの定義

classの代わりにinterfaceという予約語を使う。

```
修飾子 interface インターフェイス名 {  
    型名 フィールド名 = 式;  
    ...  
}
```

## インターフェイスの特徴

1. フィールドは、`public static final`がついた定数のみ宣言が可能である。  
修飾子は省略が可能で、省略した場合は暗黙的に`public static final`が付与される。
2. インターフェイスのメソッドは抽象メソッド、デフォルトメソッド、`static`メソッドの3種類で定義できる。(Java SE8 から)
3. インターフェイスからはオブジェクトを生成できない。



## インターフェイスの特徴:フィールド

インターフェイスのフィールドはすべて定数として扱われる。  
そのため、宣言と同時に値を代入し、初期化が必要。

```
修飾子 interface インターフェイス名 {  
    型名 フィールド名 = 式; ← 必ず初期化する  
    ...  
}
```

## インターフェイスの特徴: メソッド

メソッドにはpublic修飾子しか付与できない。  
private修飾子やprotected修飾子はコンパイルエラーとなる。

private void walk(); ← コンパイルエラー

## インターフェイスの特徴: 抽象メソッド

修飾子を記述せず「void メソッド名();」と記述すると、  
抽象メソッドと自動的に判断される。

暗黙的にpublic abstract修飾子が付与される。

void walk(); ← public abstract修飾子が付与

## インターフェイスの特徴: デフォルトメソッド

具体的な処理を記述したメソッド(デフォルトメソッド)が  
インターフェイス内で定義できる。(Java SE8以降)

```
修飾子 default 戻り値の型 メソッド名(引数リスト) {  
    処理  
}
```

## インターフェイスの特徴: staticメソッド

staticメソッドもインターフェイス内で定義できる。  
(Java SE8以降)

```
修飾子 static 戻り値の型 メソッド名(引数リスト){  
    処理  
}
```

## インターフェイスの実装

インターフェイスの実装：

インターフェイスをクラスで利用すること。

インターフェイスからはオブジェクトを生成できない。  
インターフェイスに定義されたメンバを利用するには、  
インターフェイス内の抽象メソッドをオーバーライドした  
クラス(実装クラス)を定義する必要がある。

## インターフェイスの実装

実装クラスの定義には、implementsキーワードを使用する。

```
修飾子 class クラス名 implements インターフェイス名 {  
    処理  
}
```

## インターフェイスの実装

implementsキーワードの後ろには、  
1つ以上のインターフェイスを複数指定できる。  
複数指定する場合は、「,」で区切る。

```
修飾子 class クラス名 implements インターフェイス名_インタ  
ーフェイス名 {  
    処理  
}
```



# 【Sample2201 インターフェイスの実装】 を作成しましょう



## Sample2201のポイント

インターフェイスとして定義:

BarkingAnimal2201、FourLeggedAnimal2201

インターフェイスの実装クラスとして定義:

Dog2201クラス、Cat2201クラス

## Sample2201のポイント

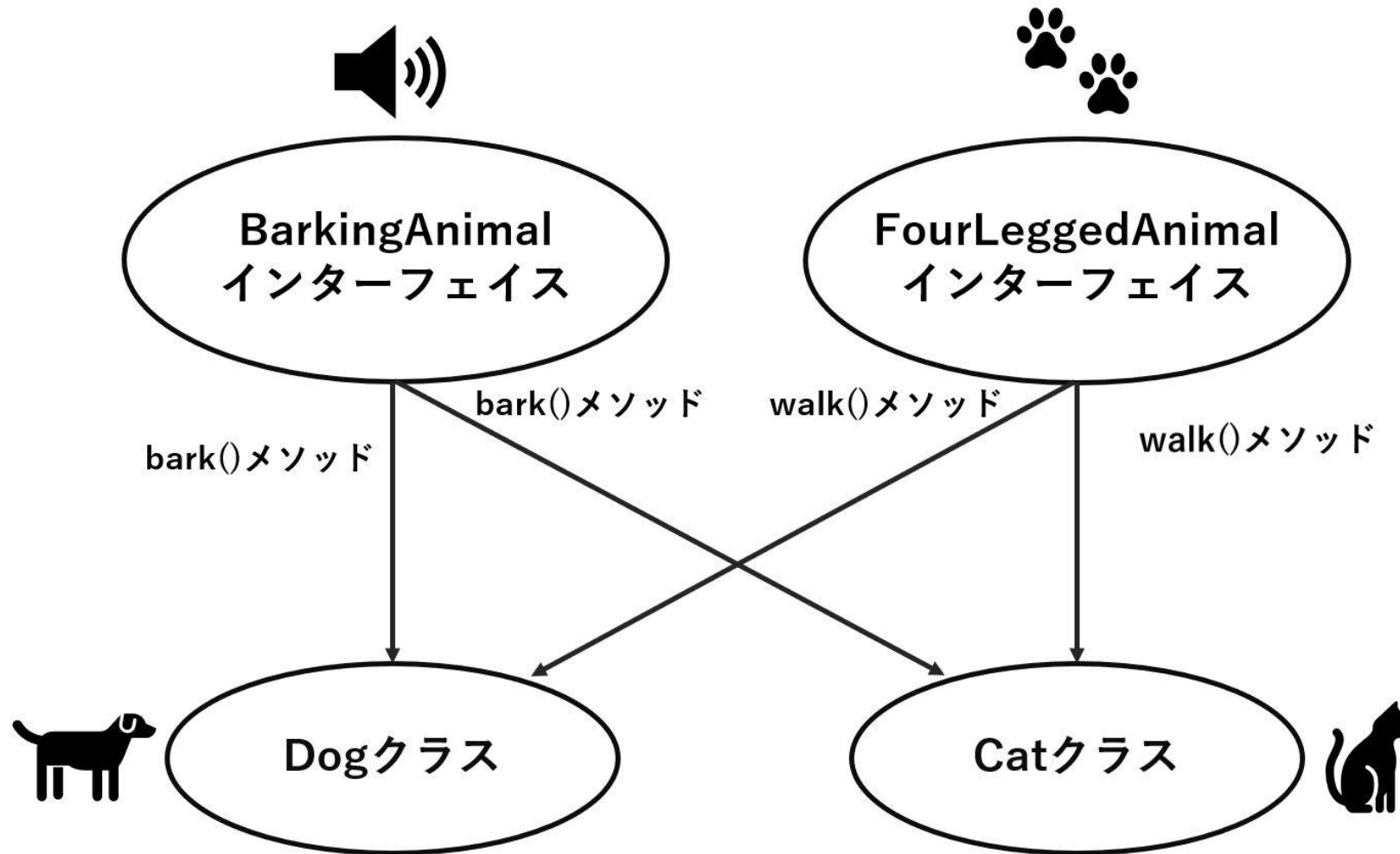
インターフェイスの実装クラスは、抽象メソッドを必ずオーバーライドする必要がある。

そのため、各クラスにはbark()メソッドとwalk()メソッドが具象メソッドとして定義されている。

```
public void bark() {  
    System.out.println("ニャーニャー");  
}
```

```
public void walk() {  
    System.out.println("猫は歩きました");  
}
```

## Sample2201のポイント




## 多重継承の仕組み

多重継承:

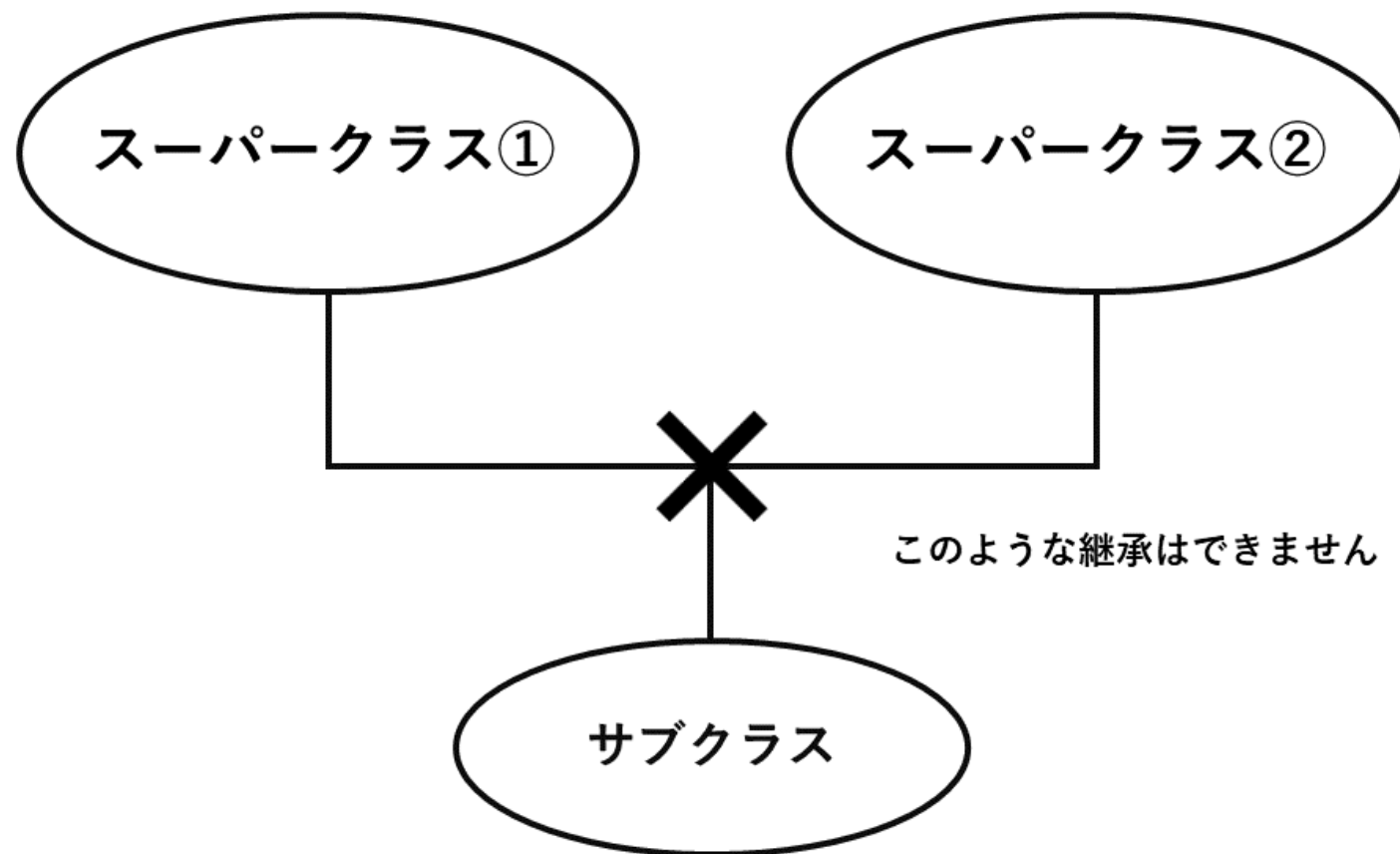
ひとつのサブクラスが複数のスーパークラスを継承すること。

クラスの多重継承は定義できない。

```
class サブクラス名 extends スーパークラス名①, スーパークラ  
ス名② {  
    ...  
}
```




## 多重継承の仕組み



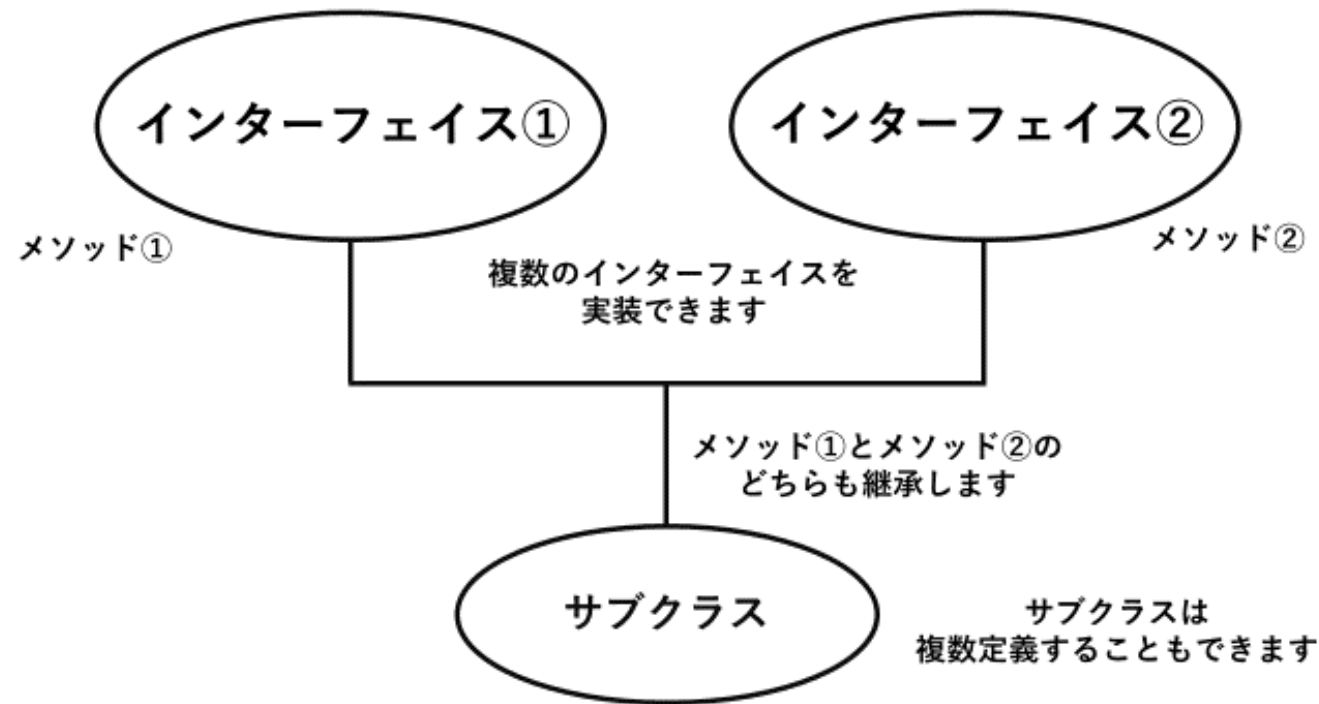
## 多重継承の仕組み

インターフェイスは多重継承できる。

```
class クラス名 implements インターフェイス名①, インターフェ  
イス名② ... {  
    ...  
}
```



## 多重継承の仕組み



クラスの多重継承できません。  
インターフェイスは多重継承できます。



## 抽象クラスとインターフェイスの違い

	抽象クラス	インターフェイス
アクセス修飾子	<code>public</code> <code>protected</code>	<code>public</code>
フィールドの定義	定義できる。	定数( <code>public static final</code> )のみ定義できる。
継承	多重継承できない。	多重継承できる。
メソッドの定義	具体的な処理(具象メソッド)も記述できる。	Java SE7までは抽象メソッドしか定義できない。 Java SE8からは <code>default</code> メソッドやクラス・メソッドを使って処理も記述できる。

## 抽象クラスとインターフェイスの違い

抽象クラス：

複数のクラスの品質を均一化したい、  
かつ複数のクラスで共通のメソッド、フィールドを使用したい場合

インターフェイス：

「複数のクラスの品質を均一化したい」、  
「複数のテンプレート(雛形)を使用して、  
機能を拡張できるようにしたい」場合

## インターフェイスの拡張

既存のインターフェイスを継承(拡張)して  
インターフェイスを定義できる。

継承: スーパーインターフェイス

「, (カンマ)」区切りで複数指定できる。

継承先: サブインターフェイス

## インターフェイスの拡張

スーパーインターフェイスを継承したいときは  
extendsを使用する。

```
サブインターフェイス名 extends スーパーインターフェイス名①  
, スーパーインターフェイス名② ... {  
    ...  
}
```

## インターフェイスの拡張

(例) Walkインターフェイスを拡張して  
FourLeggedAnimalインターフェイスを定義する

```
interface Walk {  
    ...  
}
```



スーパーインターフェイス

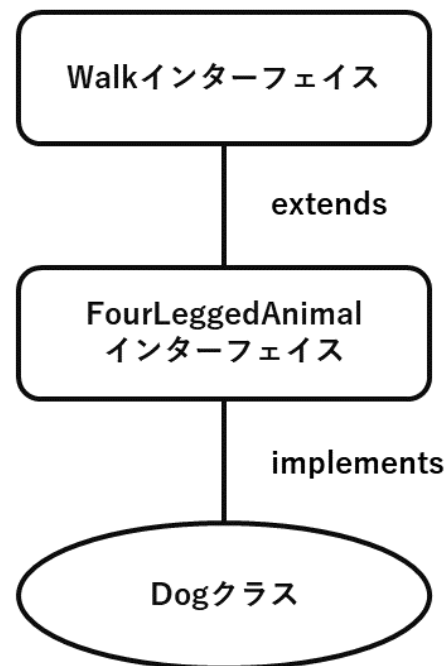
```
interface FourLeggedAnimal extends Walk {  
}
```



サブインターフェイス

## インターフェイスの拡張

DogクラスがFourLeggedAnimalインターフェイスを実装した場合、DogクラスはWalkインターフェイスのメソッドもオーバーライドする必要がある。



## 章のまとめ

- インターフェイスのフィールドは、定数となります。
- インターフェイスのメソッドは、  
処理を定義することができない抽象メソッドとなります。
- インターフェイスのオブジェクトを作成することはできません。
- SE8からは、インターフェイスにデフォルトメソッドと  
staticメソッドを定義することができます。
- インターフェイスを拡張し、  
サブインターフェイスを定義することができます。
- instanceof演算子を使うと、左辺のオブジェクトが  
右辺のクラスのものかどうかを調べることができます。