Javaプログラミング実習

21. クラスの継承

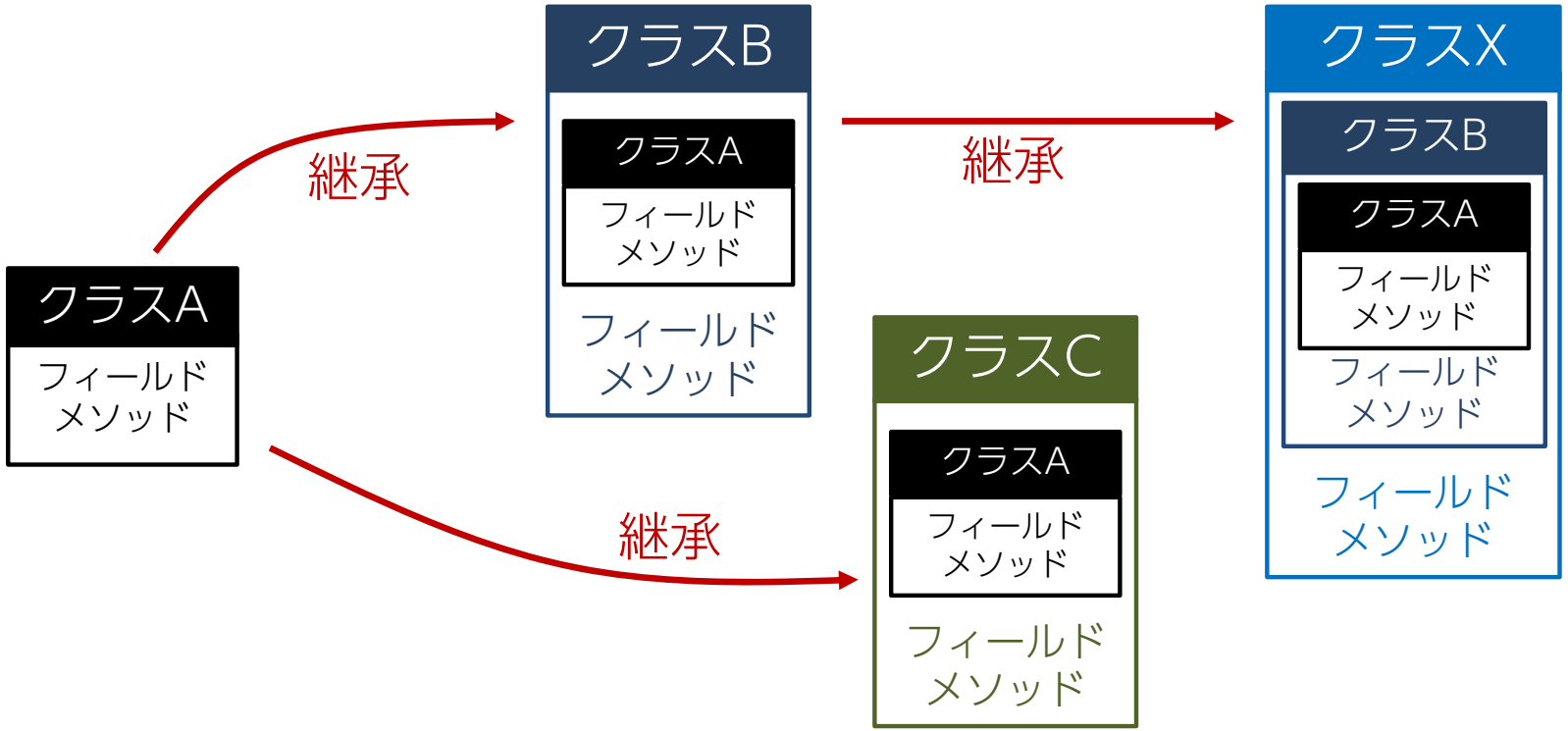
株式会社ジードライブ

今回学ぶこと

- クラスの継承とは
- メソッドのオーバーライド
- instanceof演算子
- Objectクラス
- デフォルトコンストラクタ
- finalキーワード

クラスの継承とは

- あるクラスの定義(フィールド／メソッド)を受け継ぎつつ、新たなフィールドやメソッドを追加したり既存のメソッドを変更した新たなクラスを定義すること
  - インヘリタンス(Inheritance) とも呼ばれる



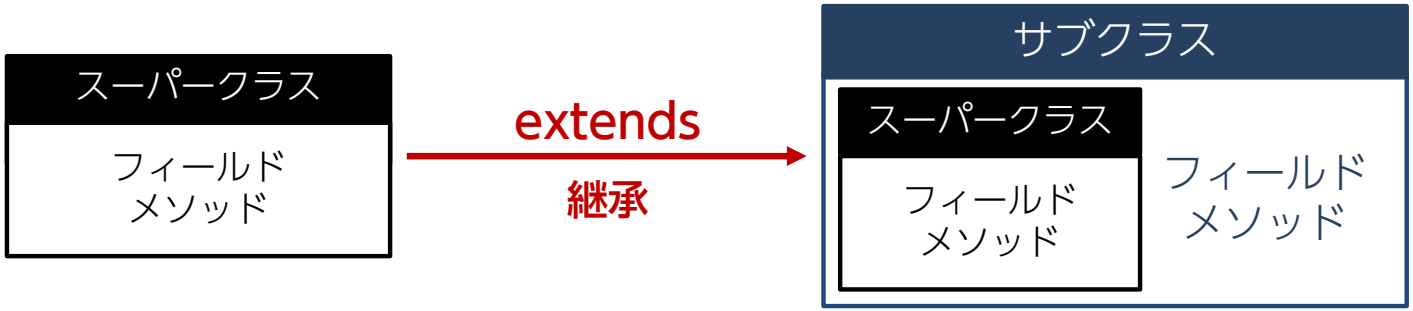
サブクラスとスーパークラス

- 継承には親子関係があり、継承元のクラスをスーパークラス(親クラス)という
- あるクラスを継承して作られたクラスをサブクラス(子クラス)という
  - サブクラスはスーパークラスのフィールドとメソッドを内包する

書式

```
public class サブクラス名 extends スーパークラス名 {

}
```



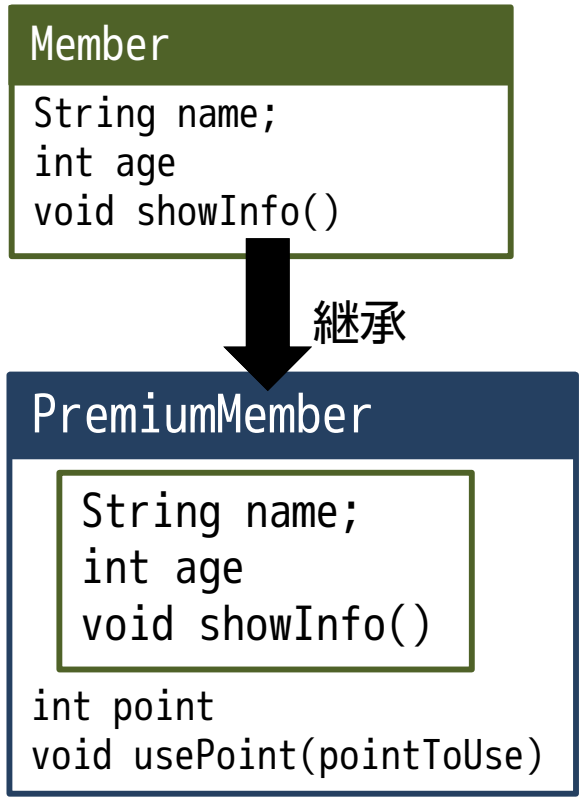
継承の例

例：Memberクラスを継承するPremiumMemberクラス

```
public class Member {
  public String name;
  public int age;
  public void showInfo() { … }
}
```

継承

```
public class PremiumMember extends Member {
  public int point;
  public void usePoint(int pointToUse) { … }
}
```

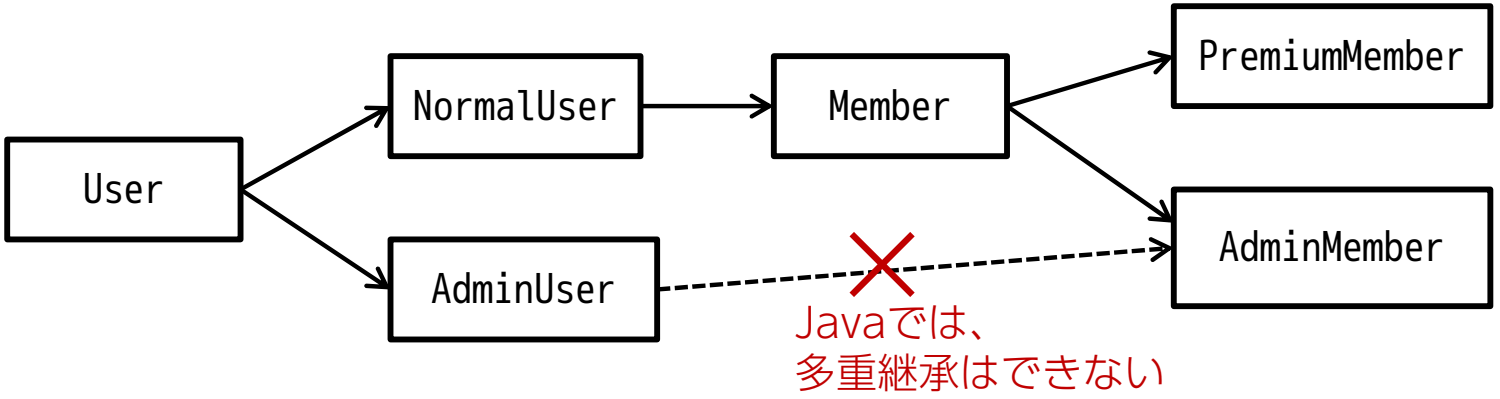


- PremiumMemberクラスには、name, ageフィールドとshowInfo()メソッドが暗黙的に内包されている
- その上でPremiumMemberクラス固有のフィールドやメソッドを追加されている

継承の注意点

- あるクラスが継承できるクラスは１つのみ
  - 複数のスーパークラス(親クラス)をもつことはできないが、複数のサブクラス(子クラス)をもつことはできる
  - 複数のクラスを同時に継承することを多重継承と呼び、Java以外の言語では多重継承ができるものもある
- ただし「あるクラスを継承したクラス」を継承したクラスを定義することは可能

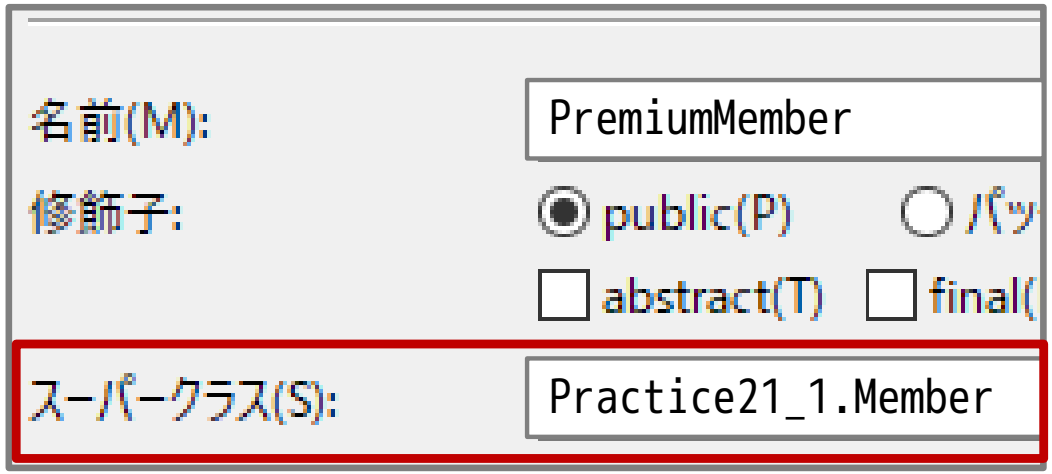
  例：Userクラスを継承したMemberクラスがあり、Memberクラスを継承したPremiumMemberクラスを定義する、等



Eclipseでの継承の指定

- 新規クラス作成時に「スーパークラス」の欄で継承元となるクラスを指定する

例：Memberを継承したPremiumMemberクラスの作成



練習

- 練習21-1

継承とアクセス修飾子

- `protected` はサブクラスからのアクセスを許可する

アクセス修飾子	説明
`public`	どのクラスからもアクセス可能
`protected`	自クラス、サブクラス、同パッケージ内のクラスからアクセス可能
アクセス修飾子なし (package-private)	自クラス、同パッケージ内のクラスからのみアクセス可能
`private`	自クラスからのみアクセス可能

private/protected

例：Memberクラスを継承するPremiumMemberクラス

```
public class Member {
  private String name;
  protected void showInfo() { … }
}
```

継承

```
public class PremiumMember extends Member {
  public void changeName(String newName) {
    name = newName;
    showInfo();
  }
}
```

nameは、privateなので利用不可
→ コンパイルエラーになる

showInfoは、protectedなので利用可能

練習

- 練習21-2

メソッドのオーバーライド

- サブクラス内で、スーパークラスで定義されているメソッドを再定義することをオーバーライドと呼ぶ
  - **@Override** というアノテーション(関連情報)を添えて、再定義を行う (@Overrideを記述することで、コンパイル時にチェックを行ってくれる)
  - 似たものにオーバーロードがあるが、**オーバーロード**は、同一クラス内で引数の型や数を変えて、同名メソッドを再定義することを指す
- この仕組みにより、サブクラス側でスーパークラスの動作を上書きしてカスタマイズすることができる

メソッドのオーバーライド

例：showInfo()メソッドのオーバーライド

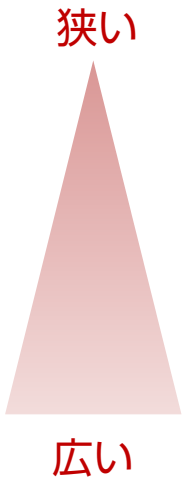
```
public class Member {
  protected void showInfo() {
    System.out.println("氏名:" + name);
    System.out.println("年齢:" + age + "歳");
  }
  …
```

継承

```
public class PremiumMember extends Member {
  @Override
  public void showInfo() {
    System.out.println("氏名:" + name);
    System.out.println("年齢:" + age + "歳");
    System.out.println("有効ポイント:" + point);
  }
  …
```

メソッドのオーバーライド

- メソッドをオーバーライドする場合、アクセス修飾子は公開範囲を広げる方向であれば変更可能
  - protectedのメソッドをオーバライドしてpublicなメソッドにすることができる(逆は不可)

アクセス修飾子	公開範囲
private package-private protected public	

スーパークラスへのアクセス

- メソッドをオーバーライドする際、親クラスのフィールドやメソッドを明示的に利用したい場合には、**super**というキーワードを使う

例

```
public class PremiumMember extends Member {
  @Override
  public void showInfo() {
    super.showInfo(); // 親クラスのshowInfo()を呼び出す
    System.out.println("有効ポイント:" + point);
  }
}
```

練習

- 練習21-3

型の互換性

- サブクラスのインスタンスは、スーパークラスの型の変数に代入することができる
  - ただし、このようにすると、サブクラスだけで定義されているフィールドやメソッドは参照できない
  - オーバーライドしたメソッドは、サブクラス定義されたものが実行される

```
Member member = new PremiumMember();
```

変数の型：
どのメソッドやフィールドが
使えるかに影響する

インスタンスの型：
実行されるメソッドの内容に影響する

直接の親の型でなくても、先祖の型であれば代入可能

型の互換性

利用例1

PremiumMemberとSeniorMemberを
それぞれの継承元Member型の配列としてまとめる

```
Member[] members = {
      new PremiumMember("山田太郎", 25, 1000),
      new SeniorMember("鈴木次郎", 57)
};
```

利用例2

メソッドの第1引数がMember型
⇒ PremiumMemberやSeniorMember型のオブジェクトを第1引数として指定できる

```
public void setNewName(Member member, String newName) {
      member.setName(newName);
};
```

練習

- 練習21-4
- 練習21-5

Object クラス

- ObjectクラスはJavaの全クラスの継承関係において頂点に位置するクラス
  - クラス定義時に extends の記述がない場合であっても、暗黙的に Object を継承していることになる

Objectクラスの主なメソッド

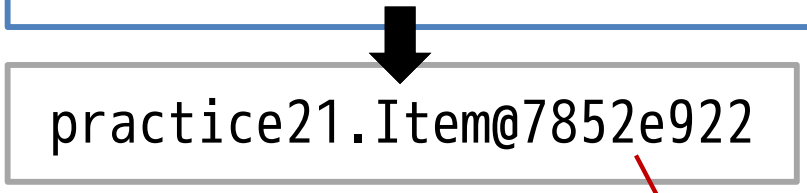
修飾子と型	メソッド	説明
public String	toString()	オブジェクトの文字列表現を返す
public boolean	equals(Object obj)	このオブジェクトが他のオブジェクトと等価かどうかを返す
public int	hashCode()	オブジェクトのハッシュコード値を返す
protected Object	clone()	このオブジェクトのコピーを作成して返す

toString()メソッド

- 文字列が期待される場所にオブジェクトがある場合に自動的に呼び出されるメソッド
  - デフォルトの戻り値は「クラス名@ハッシュコード」
  - オーバーライドすることで、任意の文字列を返すことができる

例：printlnの引数として、文字列が期待される場面

```
Item item1 = new Item("リンゴ", 100);
System.out.println(item1); // item1.toString()が呼び出される
```



ハッシュコード：各インスタンスごとの16進数値

toString()のオーバーライド

例：ItemクラスにおけるtoString()のオーバーライド

Item.java

```
public class Item {
  … フィールド／コンストラクタは省略
  @Override
  public String toString() {
    return "商品名：" + name + "／価格：" + price + "円";
  }
}
```

Eclipseではメニューからオーバーライド可能
ソース ≫ toStringの生成

Itemクラスの利用

```
Item item1 = new Item("リンゴ", 100);
System.out.println(item1);
```

⬇

商品名：りんご／価格：100円

equals()メソッド

- 引数である他のオブジェクトと同値かどうかを調べるメソッド
  - 何をもって同値とするかはオブジェクトによる
    例：クラス型が等しければ同値とする
    　　型とフィールドの値が等しければ同値とする
- Stringオブジェクト(文字列)の比較でよく使われる

```
String name1 = "Taro";
String name2 = "Jiro";
if (name1.equals(name2)) {
  System.out.println("同じ名前です");
}
else {
  System.out.println("違う名前です");
}
```

instanceof 演算子

- あるオブジェクトがあるクラスのインスタンスかどうか true / false で返す

例

```
if (m1 instanceof Member) {
  System.out.println("m1はMemberオブジェクトです");
}
else if (m1 instanceof PremiumMember) {
  System.out.println("m1はPermiumMemberオブジェクトです");
}
```

練習

- 練習21-6
- 練習21-7

デフォルトコンストラクタ

- クラス定義時にコンストラクタを省略すると暗黙的に定義される
- アクセス修飾子がpublicで、引数を持たないコンストラクタ (以下のように定義される)

```
public クラス名() {
  super();
}
```

スーパークラスのデフォルトコンストラクタを呼び出している

引数を持つコンストラクタ

- スーパークラスに引数を持つコンストラクタのみが定義されている場合、サブクラス内では明示的にコンストラクタを定義する必要がある
  - 定義内では、スーパークラスのコンストラクタを呼び出さなければいけない

スーパークラス

```
public class Parent {
  private String name;

  public Parent(String name) {
    this.name = name;
  }
}
```

引数を持つコンストラクタのみが定義されている
⇒引数なしのコンストラクタが未定義

サブクラス

```
public class Child extends Parent {
  public Child(String name) {
    super(name);
  }
}
```

親クラスのコンストラクタを呼び出し

明示的にコンストラクタを定義

finalキーワード

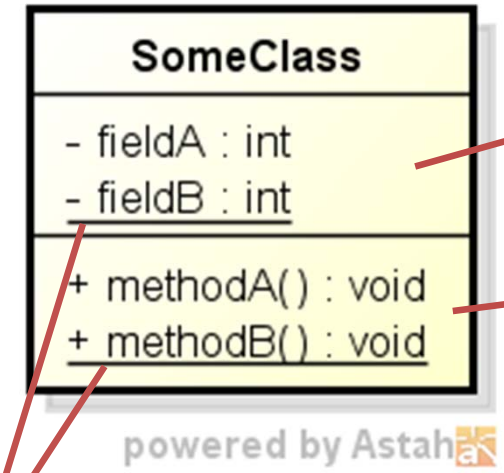
- 変数に**final**を付けると、その変数の値は変更できなくなる(定数になる)
- メソッドに**final**を付けると、そのメソッドはサブクラスでオーバーライドできなくなる
- クラスに**final**を付けると、そのクラスは継承できなくなる

練習

- 練習21-8

クラス図の書式

- クラス図における属性(フィールド)と操作(メソッド)の書式は以下のようになる
  - アクセス修飾子は**可視性**として表現する



全体に下線を引くと
static を表す

属性の書式

可視性 属性名 ： 型

操作の書式

可視性 操作名(引数 ： 型) ： 戻り値の型

可視性の記号	意味 (Javaの場合)
+	public
-	private
#	protected
~	Package (package private)

クラス図での継承の書式

- 継承はis-aの関係で行われることが多く、継承が進むにつれ、より具体的なクラスになる
  - 例：PremiumMember is a Member (プレミアム会員は会員の一種)
- 逆に継承元を辿っていくと、より抽象的／汎用的になる
  - サブクラスに共通する属性や操作をもつようになる
  - これを**汎化**と呼び、クラス図では汎化の矢印で継承を示す

例：PremiumMemberクラスがMemberクラスを継承する