クラスの継承

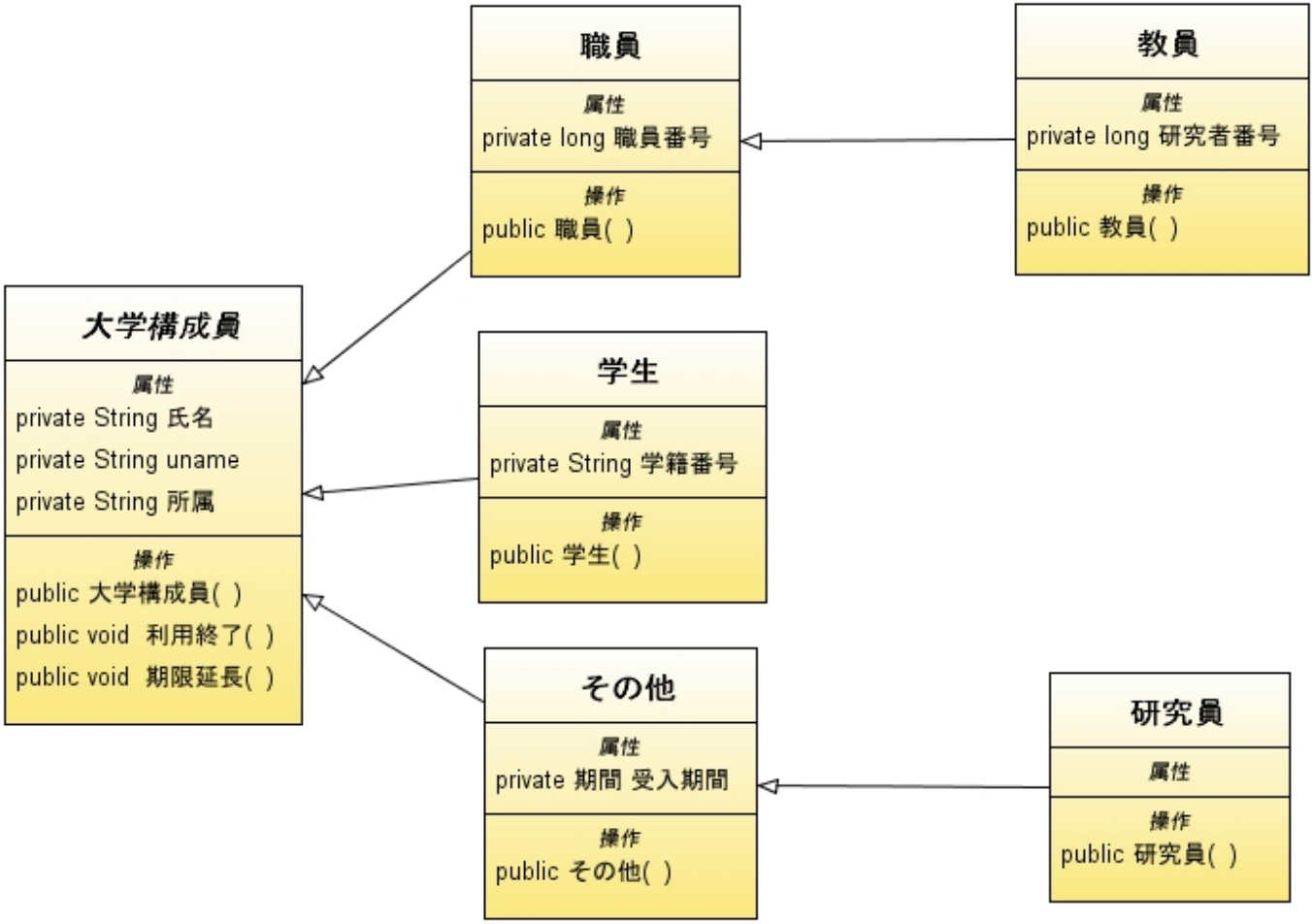
オブジェクト指向プログラミング特論

只木進一：総合情報基盤センター

Class, Super Class, Subclass

- クラスには階層構造がある
- 例：生き物の階層構造
- 例：組織の階層構造
- 上位のクラス：super class
- 抽象化、汎化 (Generalization)
- 下位のクラス：subclass
- 具体化 (Specialization)

クラス階層の例


職員
属性
private long 職員番号
操作
public 職員( )
教員
属性
private long 研究者番号
操作
public 教員( )
大学構成員
属性
private String 氏名
private String uname
private String 所属
操作
public 大学構成員( )
public void 利用終了( )
public void 期限延長( )
学生
属性
private String 学籍番号
操作
public 学生( )
その他
属性
private 期間 受入期間
操作
public その他( )
研究員
属性
操作
public 研究員( )

継承 (Inheritance)

- 全てのフィールドとメソッドが継承される
- 汎化 (Generalization)
- 共通なフィールド・メソッドの抽出
- 具体化 (Specialization)
- フィールド・メソッドの追加
- 実装の追加・変更

継承クラス

- 既存のクラスを継承して新たにクラスを定義
- 親クラスの具体化、差の導入
- 既存のクラスの再利用・拡張
- fieldの追加
- methodの追加・上書き

Method Overload

- メソッドの二つの要素
- Contact またはsignature
- メソッド名
- メソッドの引数並び
- 実装
- メソッドを多重に定義できる

Polymorphism

- 継承したクラスでメソッドを上書き
- superを使って、親のクラスを示す

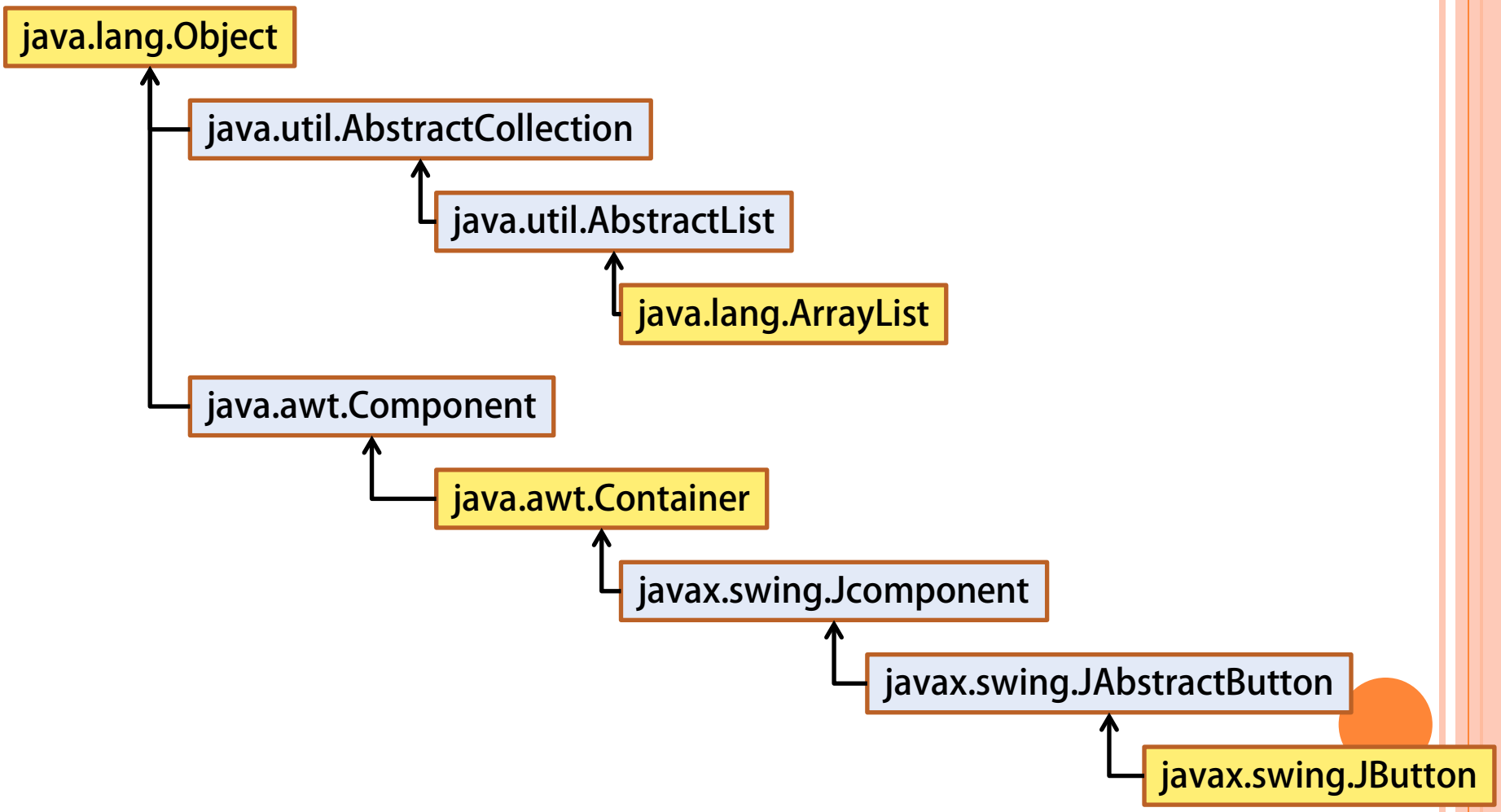
Javaでの継承の制約

- 多重継承の困難さ
- 複数の親クラスのどの性質を引き継ぐか
- Javaでは
- 一つのクラスしかextendsで継承できない
- 特殊なクラスinterface
- 複数のinterfaceを継承できる

Abstract(抽象)クラス

- 継承する元を定義する
- 共通的動作・fieldを定義する
- 一つ以上のmethodが実装されていない
- abstract メソッド
- 継承クラスで必ず実装

Classの継承例


java.lang.Object
java.util.AbstractCollection
java.util.AbstractList
java.lang.ArrayList
java.awt.Component
java.awt.Container
javax.swing.Jcomponent
javax.swing.JAbstractButton
javax.swing.JButton

INTERFACE

- 特殊なクラス
- フィールドの制限
- static finalのみ持つことができる
- これらキーワードは省略化
- メソッドの制限
- abstractメソッドのみ：実装なし
- 継承クラスで、interfaceをimplementsする
- メソッドを必ず実装する

INTERFACEの例

- java.lang.Runnable
- スレッド呼出
- java.lang.Comparable
- 比較
- java.awt.event.MouseListener
- マウスボタン操作
- java.awt.event.MouseMotionListener
- マウス動作

INTERFACEの使い方：例：RUNNABLE

```
public class B{
   private Thread t;

   public void someMethod(){
      a = new A();
      a.start();
      t = new Thread(a);
      t.start();
   }
}
```

```
public class Thread implements Runnable{
   private Runnable target;

   public void start(){初期化}

   public void run(){
      if (target != null){
         target.run();
      }
   }
}
```

```
public class A implements Runnable{
   private volatile boolean running=false;
   public void run(){
      while(running){
        処理
        if(条件)running = false;
      }
   }

   public void start(){
      running=true;
   }
}
```