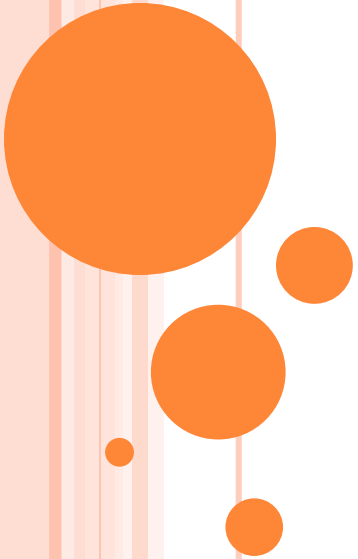


クラスの継承

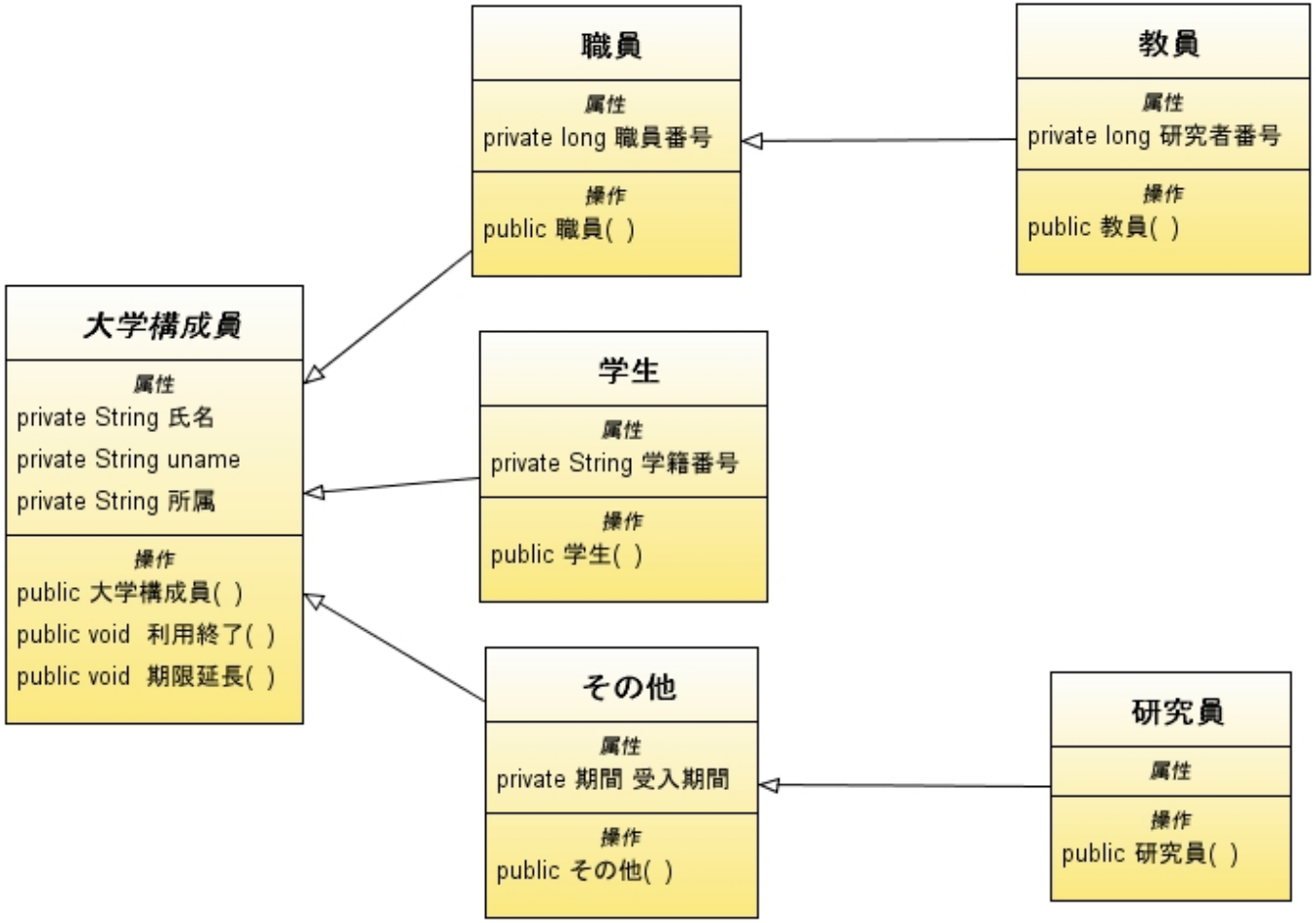
オブジェクト指向プログラミング特論

只木進一：工学系研究科

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

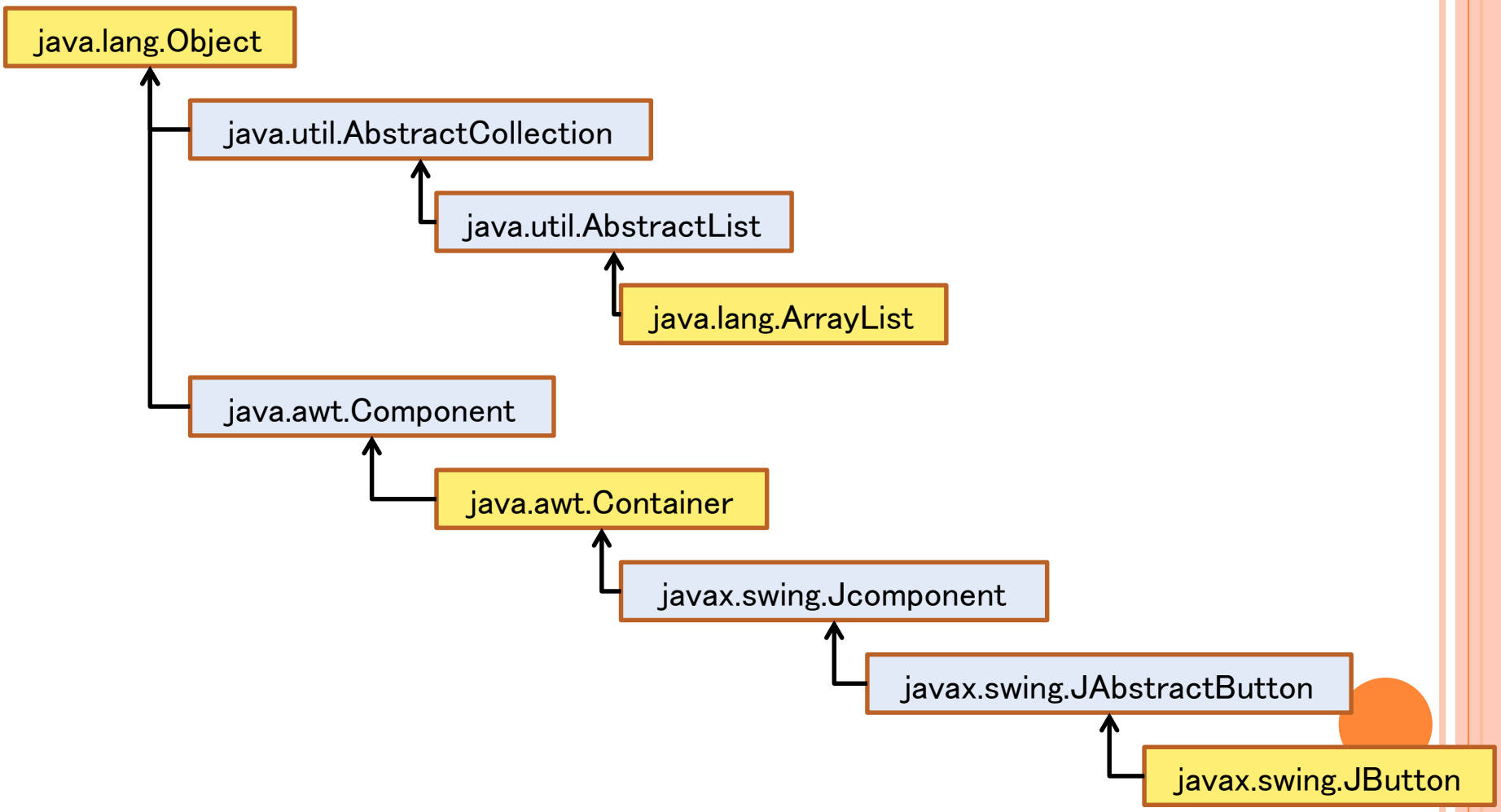
- 多重継承の困難さ
  - 複数の親クラスのどの性質を引き継ぐか
- Javaでは
  - 一つのクラスしかextendsで継承できない
  - 特殊なクラスinterface
  - 複数のinterfaceを継承できる
- Abstract classとInterfaceの使い分け

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

- クラスの側からみた階層構造
  - クラスの共通部分の抽出

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

- 操作する側の視点での抽象化
  - 同じ方法で操作できる

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

```
package thread;

public class Runner implements
Runnable{
    private volatile boolean running=false;

    private int n=0;
    @Override
    public void run() {
        while(running){
            System.out.print(n);
            System.out.print(" ");
            n++;
            try{
                Thread.sleep(1);
            }catch(InterruptedException ex){}
        }
    }
    public void start(){ running=true; }
    public void stop(){running=false;}

    static public void main(String args[]){
        Runner r=new Runner();
        r.start();
        Thread thread = new Thread(r);
        thread.start();
        for(int i=0;i<10000000;i++){
            double j=i*i+Math.random();
        }
        r.stop();
    }
}
```

LAMBDA式とSTREAMAPI

- Java8の新しい機能
- Lambda式：関数を引数として渡す
  - 従来は、クラスインスタンスとしてnewで生成していた
  - 簡素な形式に

- ListやSetなどCollectionsのstreamSPI
  - 各要素に処理を行う
  - 処理内容をLambda式で表す

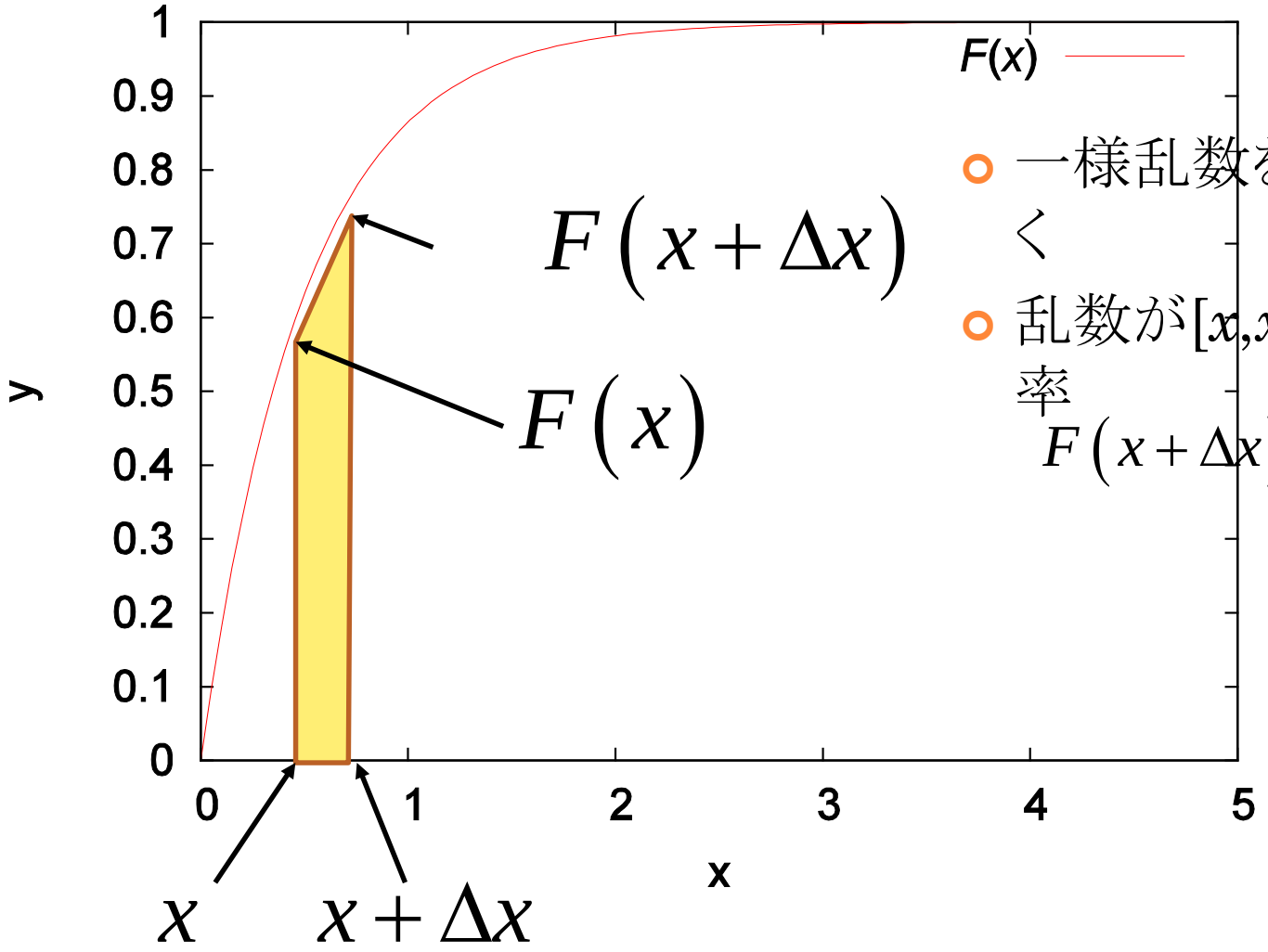
抽象クラスの例

- 乱数生成
  - 同じメソッドdouble next()で次の乱数を得る
  - 異なる分布
- 共通的クラスMyRandom
  - 一様乱数 Uniform
  - 指数分布 Exponential
  - 正規分布 Gaussian
  - 区間の相対頻度定義 Step

変換法による乱数生成

- 確率密度 $f(x)$ に従う乱数
  - 正規化条件 $$\int_{\min}^{\max} f(x)\mathrm{d}x = 1$$
  - 確率分布 $$0 \le F(x) = \int_{\min}^{x} f(x)\mathrm{d}x \le 1$$
  - 一様乱数 $0 \le r < 1$ より

$$x = F^{-1}(r)$$



- 一様乱数をy軸にばらまく
- 乱数が$[x,x+\Delta x)$に入る確率

$$F\left(x+\Delta x\right)-F\left(x\right)\simeq f\left(x\right)\Delta x$$

例:指数分布

$$f(x)=\frac{1}{a}e^{-ax}$$

$$F(x)=\int_0^x f(y)\mathrm{d}y=1-e^{-ax}$$

- 一様乱数　$r\in[0,1)$

$$x=F^{-1}(r)$$

特殊な例：正規分布

$$f(x)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

- 2次元正規分布を極座標で表す

$$\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{x^2+y^2}{2\sigma^2}\right]\mathrm{d}x\mathrm{d}y=\frac{1}{\sqrt{2\pi\sigma^2}}r\exp\left[-\frac{r^2}{2\sigma^2}\right]\mathrm{d}r\mathrm{d}\theta$$

- 動径方向は変換法で生成可能
  - 角度方向は一様分布

- 二つの一様分布(a,b)から

$$r = \sqrt{-2\sigma^2 \ln(1-a)}$$

$$\theta = 2\pi b$$

棄却法

- 変換法で生成できない場合
  - 分布関数$f(x)$の変域$[a,b)$とし、その最大値をmとする
  - $[0,1)$の乱数の組(x,y)を生成する

$$z=(x-b)\times x+b$$

  - $y<f(z)$　ならばzを乱数として採用する
    - それ以外は棄却し、再度乱数を生成する

例：

$$f(x)=\begin{cases}\dfrac{1}{a+b(1-a)} & 0\le x<a\\[2ex] \dfrac{b}{a+b(1-a)} & a\le x<1\end{cases}$$

課題

- プログラムを実際に動かし、動作を確認しなさい。
- 動作の確認には、生成されたヒストグラムと想定していた確率密度関数を重ね合わせることで行いなさい。