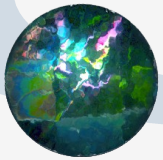
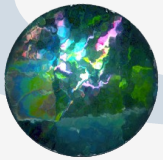


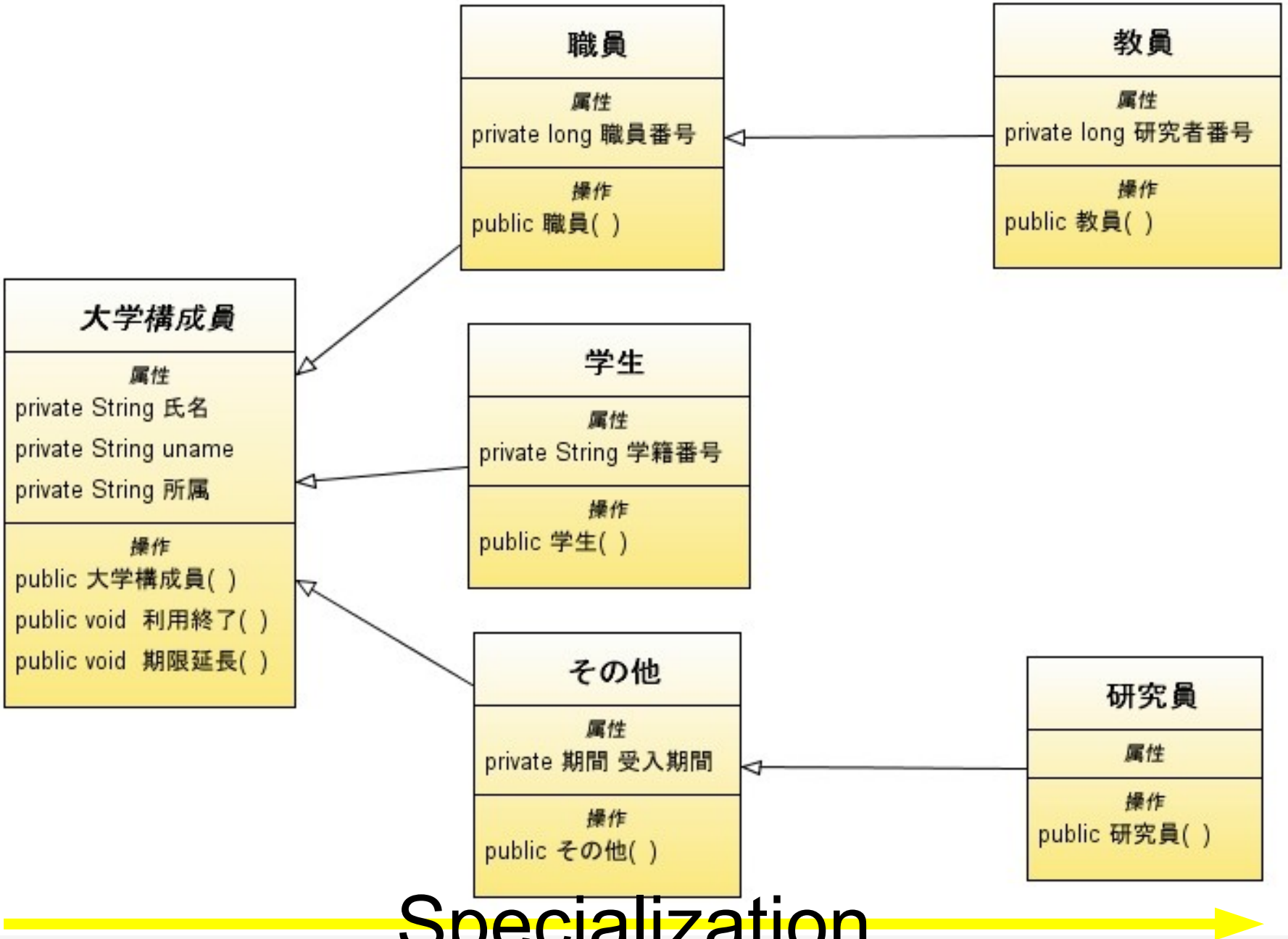
クラスとその継承



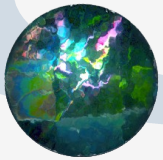
Class, Super Class, Subclass

- 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)

Generalization

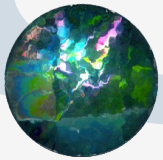


Specialization



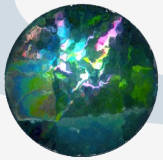
継承 (Inheritance)

- 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



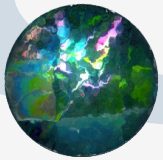
継承クラス

- 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



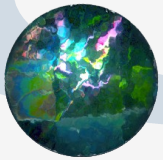
Method Overload

- 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



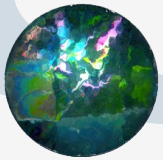
Polymorphism

- 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



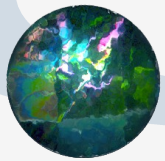
Javaでの継承の制約

- 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

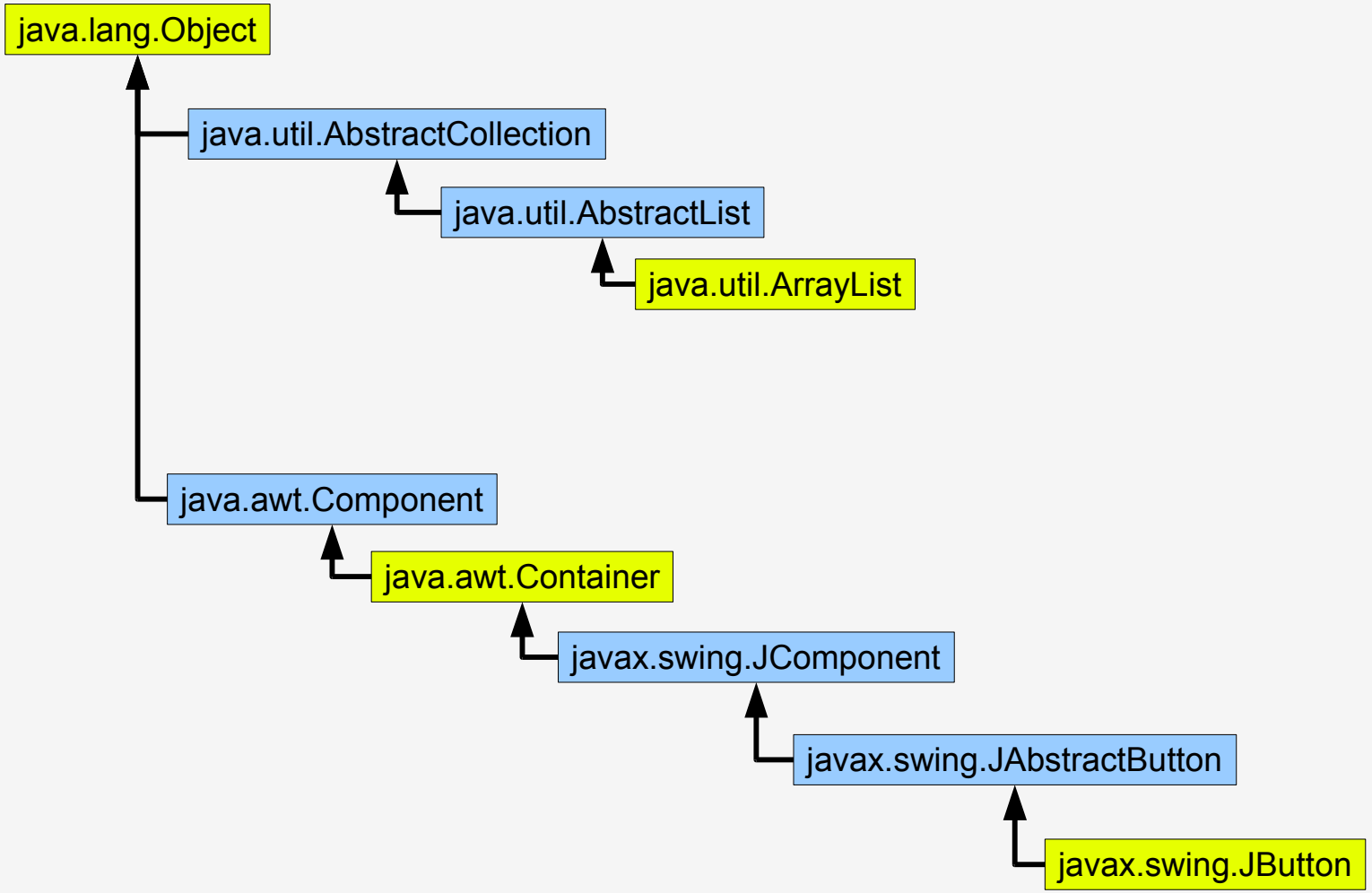


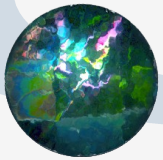
Abstract(抽象)クラス

- 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



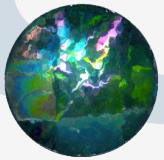
abstract classの例





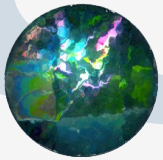
interface

- 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

- 他のクラスからの呼出の標準化
- 例：java.lang.Runnable

```
public class B{
    private Thread t;

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

```
public class A implements Runnable{

    public void run(){
      ....
    }
}
```

```
public class Thread{
    private Runnable target;

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

}
```

Member.java

```
package universityMembers;

/**
 *
 * @author tadaki
 */
abstract public class Member {

    private String name;
    private String uname;
    private String affiliation;
    private boolean valid = true;
    private Period period;

    public Member(String name, String uname, String affiliation) {
        this.name = name;
        this.uname = uname;
        this.affiliation = affiliation;
        period = new Period();
    }
    //無効化処理

    public void invalidate() {
        setValid(false);
    }
    //期限延長処理

    abstract public void extend(java.util.Calendar c);

    public Period getPeriod() {
        return period;
    }

    public void setPeriod(Period period) {
        this.period = period;
    }

    public String getName() {
        return name;
    }

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

```
    public String getUname() {
        return uname;
    }

    public String getAffiliation() {
        return affiliation;
    }

    public void setAffiliation(String affiliation) {
        this.affiliation = affiliation;
    }

    public boolean isValid() {
        return valid;
    }

    public void setValid(boolean valid) {
        this.valid = valid;
    }
}
```

Student.java

```
package universityMembers;

import java.util.Calendar;

/**
 *
 * @author tadaki
 */
public class Student extends Member {

    private String studentID = null;

    static public enum REC {

        GRAD("卒業・終了"),
        TERM("除籍"),
        MOVE("転学部・転学科"),
        NONE("");
        private String jName;

        REC(String jName) {
            this.jName = jName;
        }

        public String getJName() {
            return jName;
        }
    }
    private REC rec = REC.NONE;

    public Student(String name, String uname,
            String affiliation, String studentID) {
        super(name, uname, affiliation);
        this.studentID = studentID;
        getPeriod().getTo().add(Calendar.YEAR, 4);
    }

    @Override
    public void extend(Calendar c) {
        getPeriod().getTo().add(Calendar.YEAR, 1);
    }

    @Override
    public void invalidate() {
```

```
            invalidate(REC.GRAD);
        }

        public void invalidate(REC rec) {
            super.invalidate();
            this.rec = rec;
        }
    }
```

Period.java

```
package universityMembers;
import java.util.Calendar;
/**
 *
 * @author tadaki
 */
public class Period {
    private Calendar from;
    private Calendar to;

    public Period() {
        this.from = Calendar.getInstance();
        this.to=(Calendar)from.clone();
    }
    public Period(Calendar from, Calendar to) {
        this.from = from;
        this.to = to;
    }

    public Calendar getFrom() {
        return from;
    }

    public void setFrom(Calendar from) {
        this.from = from;
    }

    public Calendar getTo() {
        return to;
    }

    public void setTo(Calendar to) {
        this.to = to;
    }
}
```

Runner.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package thread;

/**
 *
 * @author tadaki
 */
public class Runner implements Runnable{
    private volatile boolean running=false;

    private int n=0;
    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 runner = new Thread(r);
        runner.start();
        for(int i=0;i<10000000;i++){
            double j=i*i+Math.random();
        }
        r.stop();
    }
}
```