

計算機科学第一

2013年度第6回

日付

もくじ

- ❖ Item 8: General contract
- ❖ Item 9: Override hashCode
- ❖ Item 10: Override toString
- ❖ Item 11: Override clone judiciously
- ❖ Item 12: Comparable

Chapter 3:

Methods common to all

コンストラクタの概要

[Object\(\)](#)

メソッドの概要

protected Object	clone() このオブジェクトのコピーを作成し	String	toString() オブジェクトの文字列表現を返
boolean	equals(Object obj) このオブジェクトと「等価」になる	void	wait() ほかのスレッドがこのオブジェ させます。
protected void	finalize() このオブジェクトへの参照はもうな されます。	void	wait(long timeout) 別のスレッドがこのオブジェク まで、現在のスレッドを待機させます
Class <?>	getClass() この Object の実行時クラスを返しま	void	wait(long timeout, int nanos) ほかのスレッドがこのオブジェ スレッドに割り込みをかけたり、指定
int	hashCode() オブジェクトのハッシュコード値を		
void	notify() このオブジェクトのモニターで待機中のスレッドを1つ再開します。		
void	notifyAll() このオブジェクトのモニターで待機中のすべてのスレッドを再開します。		
String	toString() オブジェクトの文字列表現を返します。		

上書きを想定したメソッド

final でないメソッド

メソッド	概要
<code>equals</code>	等価性（＝同値性）の判定
<code>hashCode</code>	ハッシュコードの計算
<code>clone</code>	コピーを作成
<code>toString</code>	文字列表現に変換

VIII: Obey the general contract
when overriding equals

VIII: equals は同値関係

同値 (equivalence/equals) と
同一 (identity/==) は違うのだ

上書きすべきでない場合

- ❖ オブジェクトの同一性が本質的
- ❖ 論理的な同値性が意味を持たない場合
- ❖ 親クラスが実装する同値性でokな場合
- ❖ privateなクラスの場合
 - ❖ 念のため例外を投げるのが吉

いつ上書きするの？

- ❖ Value class (値を実装するクラス)
 - ❖ クラスが求める同値性 \neq 同一性 \wedge
親クラスの equals に不満

どのように上書きするの？

- ❖ 同値関係 = 反射律 \wedge 対称律 \wedge 推移律
- ❖ 整合性: x, y が変化しない限り $x.equals(y)$ の結果は同一であること
- ❖ $\forall x \neq \text{null}. x.equals(\text{null}) = \text{false}$

Object.equals より

equals

```
public boolean equals(Object obj)
```

このオブジェクトと「等価」になるオブジェクトがあるかどうかを示します。

equals メソッドは、null 以外のオブジェクト参照での同値関係を実装します。

- 反射性 (*reflexive*): null 以外の参照値 x について、 $x.equals(x)$ は true を返す
- 対称性 (*symmetric*): null 以外の参照値 x と y について、 $x.equals(y)$ は、 $y.equals(x)$ が true を返す場合だけ true を返す
- 推移性 (*transitive*): null 以外の参照値 x 、 y 、 z について、 $x.equals(y)$ が true を返し、かつ $y.equals(z)$ が true を返す場合に、 $x.equals(z)$ は true を返す
- 整合性 (*consistent*): null 以外の参照値 x および y について、 $x.equals(y)$ を複数呼び出すと常に true を返すか、常に false を返す。これは、オブジェクトに対する equals による比較で使われた情報が変更されていないことが条件である
- null でない任意の参照値 x について、 $x.equals(null)$ は false を返す

同値性の復習

- ❖ 反射律: $x.equals(x) = \text{true}$
- ❖ 対称律: $x.equals(y) = y.equals(x)$
- ❖ 推移律:
$$x.equals(y) \wedge y.equals(z) \Rightarrow x.equals(z)$$

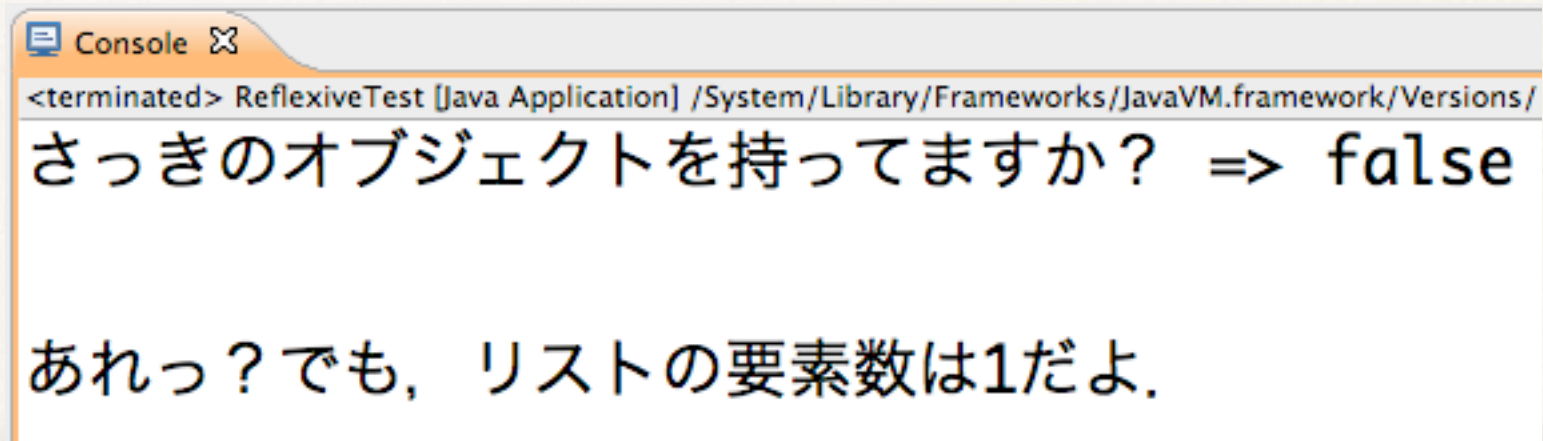
反射律を壊してみよう

```
class NotReflexive {  
    public boolean equals(Object x) {  
        if (x == this) return false;  
        return super.equals(x);  
    }  
}
```

* @author wakita

```
public class ReflexiveTest {  
    * 反射律を破壊したオブジェクトをリストに追加しても、追加はできるものの、リストから取  
    private void run() {  
        List<NotReflexive> s = vector();  
        NotReflexive x = new NotReflexive();  
        s.add(x);  
  
        System.out.printf("さっきのオブジェクトを持っていますか？ => %s\n\n", s.contains(x));  
        System.out.printf("あれっ？でも、リストの要素数は%dだよ。 \n", s.size());  
    }  
}
```

反射律を壊したら...



```
Console X
<terminated> ReflexiveTest [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/
さっきのオブジェクトを持っていますか? => false

あれっ?でも, リストの要素数は1だよ.
```

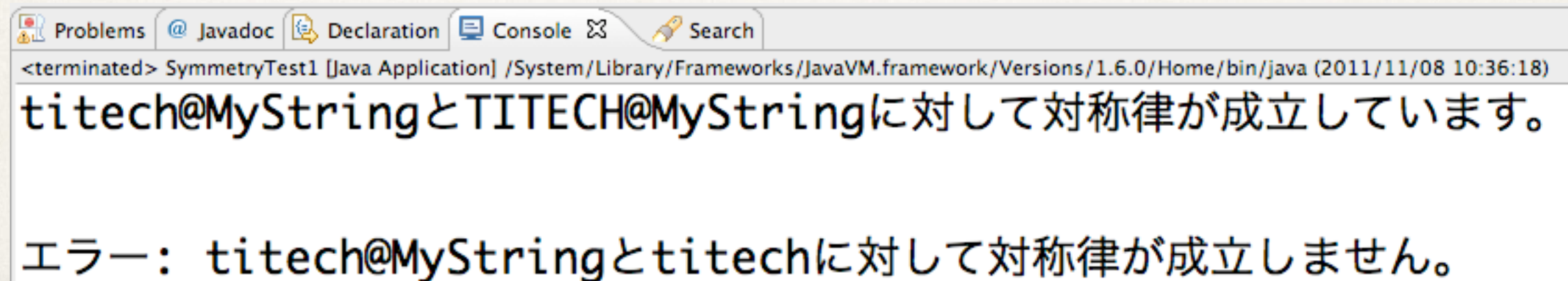

対称律を壊してみよう

```
final class MyString {  
    private final String s;  
  
    public MyString(String s) {  
        if (s == null) throw new NullPointerException();  
        this.s = s;  
    }  
  
    public boolean equals(Object o) {  
        if (o instanceof MyString)  
            return s.equalsIgnoreCase(((MyString) o).s);  
        if (o instanceof String)  
            return s.equalsIgnoreCase((String)o);  
        return false;  
    }  
}
```

テストコード

```
private void run() {  
    MyString titech = new MyString("titech"), TITECH = new MyString(  
        "TITECH");  
    String titech_s = "titech";  
    EqualityTests.symmetry(titech, TITECH);  
    EqualityTests.symmetry(titech, titech_s);  
}
```


対称律を壊したら...



The screenshot shows an IDE console window with the following content:

```
<terminated> SymmetryTest1 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 10:36:18)  
titech@MyStringとTITECH@MyStringに対して対称律が成立しています。  
  
エラー: titech@MyStringとtitechに対して対称律が成立しません。
```

それがどうした！

それがどうした！

対称律を怒らすと怖いよ

日付

対称律を壊すと 見つかるものも見つからん

```
private void test2(Object... s) {  
    List<Object> l = vector();  
    int i = 0;  
    for (Object x : s) {  
        l.add(x);  
        out.printf(" - l[%d] = %15s;  index = %d => %15s\n",  
                    i, l.get(i), l.indexOf(x), l.get(l.indexOf(x)));  
        i++;  
    }  
    out.println();  
}  
  
test2(TITECH, titech_s);  
test2(titech_s, TITECH);
```


対称律を壊すと見つかるものも見つからない

```
private void test2(Object... s) {  
    List<Object> l = vector();  
    int i = 0;  
    for (Object x : s) {  
        l.add(x);  
        out.printf(" - l[%d] = %15s; index = %d => %15s\n",  
                    i, l.get(i), l.indexOf(x), l.get(l.indexOf(x)));  
        i++;  
    }  
    out.println();  
}
```

リストのなかでxが
何番目かを返す

```
test2(TITECH, titech_s);  
test2(titech_s, TITECH);
```

確かに入っているのに...

挿入順	挿入した文字列	見つかった場所	検索結果
-----	---------	---------	------

l[0]	= TITECH@MyString;	index = 0 =>	TITECH@MyString
l[1]	= titech;	index = 1 =>	titech
l[0]	= titech;	index = 0 =>	titech
l[1]	= TITECH@MyString;	index = 0 =>	titech

確かに入っているのに...

挿入順	挿入した文字列	見つかった場所	検索結果
-----	---------	---------	------

l[0]	= TITECH@MyString;	index = 0 =>	TITECH@MyString
l[1]	= titech;	index = 1 =>	titech

l[0]	= titech;	index = 0 =>	titech
l[1]	= TITECH@MyString;	index = 0 =>	titech

推移律も同様

継承 vs equals

- ❖ 継承機構と equals は案外相性が悪い
- ❖ 事例：ColorPoint vs Point
 - ❖ Point: (x, y)
 - 同値 = 座標が同じ
 - ❖ ColorPoint: (x, y)
 - 同値 = 座標が同じ ∧ 色が同じ

Point → ColorPoint

```
class Point {  
    protected final int x, y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public boolean equals(Object o) {  
        if (!(o instanceof Point)) return false;  
        Point p = (Point) o;  
        return p.x == x && p.y == y;  
    }  
}
```

p, q が Point 同士 の とき

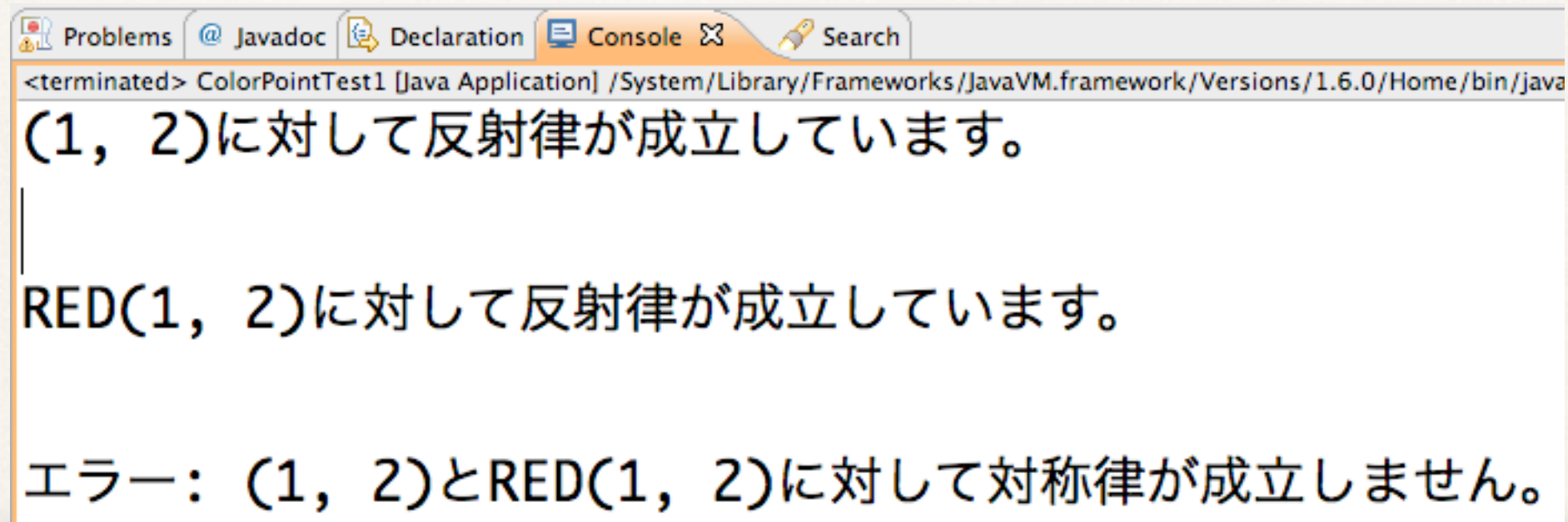
$$(x_p, y_p) = (x_q, y_q)$$

```
class ColorPoint extends Point {  
    private final Color color;  
  
    public ColorPoint(int x, int y, Color color) {  
        super(x, y);  
        this.color = color;  
    }  
  
    public boolean equals(Object o) {  
        if (!(o instanceof ColorPoint)) return false;  
        return super.equals(o) &&  
            ((ColorPoint) o).color == color;  
    }  
}
```

p, q が ColorPoint 同士 の とき

$$(x_p, y_p, c_p) = (x_q, y_q, c_q)$$

ColorPointTest1



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, Console, and Search. The console output is as follows:

```
<terminated> ColorPointTest1 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java  
(1, 2)に対して反射律が成立しています。  
|  
RED(1, 2)に対して反射律が成立しています。  
エラー: (1, 2)とRED(1, 2)に対して対称律が成立しません。
```

- $(1, 2).equals(RED(1, 2))$ — `Point.equals`
RED(1, 2) を (1,2) と見做して比較 → true
- $RED(1, 2).equals((1, 2))$ — `ColorPoint.equals`
クラスが異なるので false

Point → ColorPoint

改良してみた

```
class Point {
    protected final int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public boolean equals(Object o) {
        if (!(o instanceof Point)) return false;
        Point p = (Point) o;
        return p.x == x && p.y == y;
    }
}
```

p, q がPoint同士の時
 $(x_p, y_p) = (x_q, y_q)$

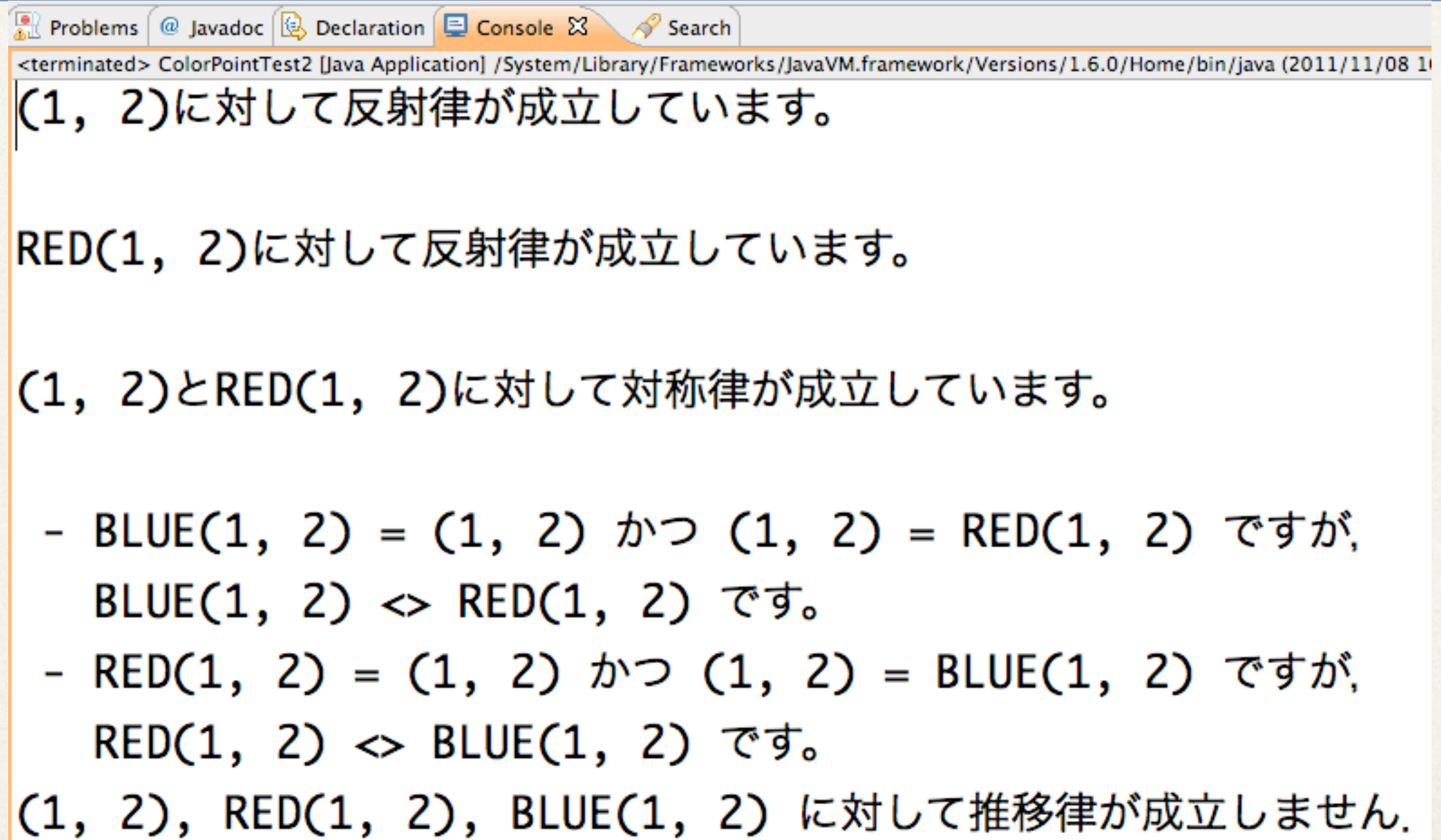
```
class ColorPoint extends Point {
    private final Color color;

    public ColorPoint(int x, int y, Color color) {
        super(x, y);
        this.color = color;
    }

    public boolean equals(Object o) {
        if (!(o instanceof ColorPoint))
            return o.equals(this);
        return super.equals(o) &&
            color == ((ColorPoint) o).color;
    }
}
```

- 引数(o)が ColorPoint なら
 $(x_p, y_p) = (x_q, y_q) \ \&\& \ c_p == c_q$
- そうでなかったら
 $o.equals(this)$

テスト



The screenshot shows an IDE window titled "ColorPointTest2 [Java Application]". The console tab is active, displaying the following text:

```
<terminated> ColorPointTest2 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 11:08:11)
(1, 2)に対して反射律が成立しています。

RED(1, 2)に対して反射律が成立しています。

(1, 2)とRED(1, 2)に対して対称律が成立しています。

- BLUE(1, 2) = (1, 2) かつ (1, 2) = RED(1, 2) ですが,
  BLUE(1, 2) <> RED(1, 2) です。
- RED(1, 2) = (1, 2) かつ (1, 2) = BLUE(1, 2) ですが,
  RED(1, 2) <> BLUE(1, 2) です。

(1, 2), RED(1, 2), BLUE(1, 2) に対して推移律が成立しません。
```

テスト

```
Problems @ Javadoc Declaration Console Search
<terminated> ColorPointTest2 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 11:08:11)

(1, 2)に対して反射律が成立しています。

RED(1, 2)に対して反射律が成立しています。

(1, 2)とRED(1, 2)に対して対称律が成立しています。

- BLUE(1, 2) = (1, 2) かつ (1, 2) = RED(1, 2) ですが,
  BLUE(1, 2) <> RED(1, 2) です。
- RED(1, 2) = (1, 2) かつ (1, 2) = BLUE(1, 2) ですが,
  RED(1, 2) <> BLUE(1, 2) です。

(1, 2), RED(1, 2), BLUE(1, 2) に対して推移律が成立しません。
```

クラスが異なるので, `(1, 2).equals(BLUE(1, 2))` を呼ぶ. これは
`Color.equals` → true

テスト

```
Problems Javadoc Declaration Console Search
<terminated> ColorPointTest2 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 10:10:10)

(1, 2)に対して反射律が成立しています。

RED(1, 2)に対して反射律が成立しています。

(1, 2)とRED(1, 2)に対して対称律が成立しています。

- BLUE(1, 2) = (1, 2) かつ (1, 2) = RED(1, 2) ですが,
  BLUE(1, 2) <> RED(1, 2) です。
- RED(1, 2) = (1, 2) かつ (1, 2) = BLUE(1, 2) ですが,
  RED(1, 2) <> BLUE(1, 2) です。

(1, 2), RED(1, 2), BLUE(1, 2) に対して推移律が成立しません.
```

Color.equals が (引数が ColorPoint と意識せずに) 呼ばれる → true

テスト

```
Problems @ Javadoc Declaration Console X Search
<terminated> ColorPointTest2 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 11:08:10)

(1, 2)に対して反射律が成立しています。

RED(1, 2)に対して反射律が成立しています。

(1, 2)とRED(1, 2)に対して対称律が成立しています。

- BLUE(1, 2) = (1, 2) かつ (1, 2) = RED(1, 2) ですが、
  BLUE(1, 2) <> RED(1, 2) です。
- RED(1, 2) = (1, 2) かつ (1, 2) = BLUE(1, 2) ですが、
  RED(1, 2) <> BLUE(1, 2) です。

(1, 2), RED(1, 2), BLUE(1, 2) に対して推移律が成立しません。

ColorPoint.equals が呼ばれ、引数のクラスも ColorPoint なので、色も
含めた比較を行う → false
```


以上より

- ❖ クラス継承し，値を拡張しつつ，同値性契約を満すのは困難

大事なことなので繰り返す

- ❖ クラス継承し、値を拡張しつつ、同値性契約を満すことは困難

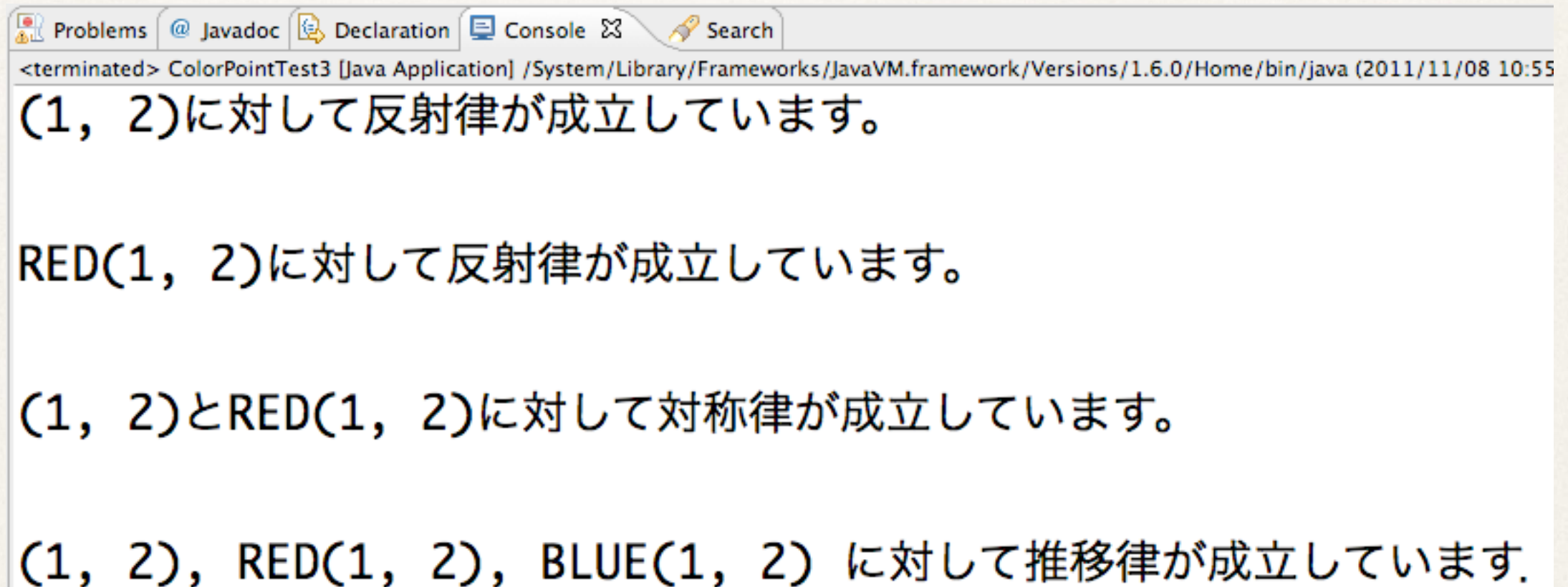
クラス継承以外の拡張法

値クラスの拡張：

継承=凶， 組み合わせ=吉

```
class ColorPoint {  
    private Point p;  
    private final Color color;  
  
    public ColorPoint(int x, int y, Color color) {  
        if (color == null) throw new NullPointerException;  
        p = new Point(x, y);  
        this.color = color;  
    }  
  
    public Point asPoint() { return p; }  
  
    public boolean equals(Object o) {  
        if (!(o instanceof ColorPoint))  
            return false;  
        ColorPoint cp = (ColorPoint)o;  
        return cp.p.equals(p) && cp.color.equals(color);  
    }  
}
```


テスト成功！



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, Console, and Search. The console output is as follows:

```
<terminated> ColorPointTest3 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 10:55)
(1, 2)に対して反射律が成立しています。


RED(1, 2)に対して反射律が成立しています。

(1, 2)とRED(1, 2)に対して対称律が成立しています。

(1, 2), RED(1, 2), BLUE(1, 2) に対して推移律が成立しています.
```


3種の実装例のまとめ

Test1	P2	CP2
PI	x, y	x, y
CPI	F	x, y, c



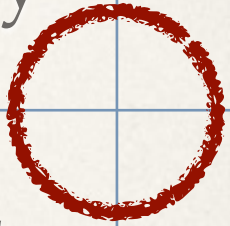
対称律が破綻

Test2	P2	CP2
PI	x, y	x, y
CPI	x, y	x, y, c



推移律が破綻

Test3	P2	CP2
PI	x, y	F
CPI	F	p, c



めでたし, めでたし

どのように上書きするの？

- ❖ 同値関係 = 反射律 \wedge 対称律 \wedge 推移律
- ❖ 整合性: x, y が変化しない限り $x.equals(y)$ の結果は同一であること
- ❖ $\forall x \neq \text{null}. x.equals(\text{null}) = \text{false}$

整合性

- ❖ equals の検査には、素性のよいフィールドを使うこと。例外》java.io.URL

```
public boolean equals(Object obj)
```

この URL と別のオブジェクトとが等しいかどうかを比較します。

指定されたオブジェクトが URL でない場合、このメソッドは直ちに `false` を返します。

2つの URL オブジェクトが等しいのは、同じプロトコルを持ち、同じホストを参照し、ホスト上のポート番号が同じで、ファイルとファイルのフラグメントが同じ場合です。

2つのホストが等価と見なされるのは、両方のホスト名が同じ IP アドレスに解決されるか、どちらかのホスト名を解決できない場合は、大文字小文字に関係なくホスト名が等しいか、両方のホスト名が `null` に等しい場合です。

ホスト比較には名前解決が必要なので、この操作はブロック操作です。

注:equals の定義された動作は、HTTP の仮想ホストと一致しないことが知られています。

整合性

- ❖ equals の検査には、素性のよいフィールドを使うこと。例外》java.io.URL

```
public boolean equals(Object obj)
```

この URL と別のオブジェクトとが等しいかどうかを比較します。

指定されたオブジェクトが URL でない場合、このメソッドは直ちに `false` を返します。

2 つの URL オブジェクトが等しいのは、同じプロトコルを持ち、同じホストを参照し、ホスト上のポート番号が同じで、ファイルとファイルのフラグメントが同じ場合です。

2 つのホストが等価と見なされるのは、両方のホスト名が同じ IP アドレスに解決されるか、どちらかのホスト名を解決できない場合は、大文字小文字に関係なくホスト名が等しいか、両方のホスト名が `null` に等しい場合です。

ホスト比較には名前解決が必要なので、この操作はブロック操作です。

注:equals の定義された動作は、HTTP の仮想ホストと一致しないことが知られています。

* java.net.URL で実装されている equals の整合性が破綻していることを示す実験.

```
private void run() {  
    try {  
        URL url1 = new URL("http://yahoo.co.jp/");  
        URL url2 = new URL("http://f11.top.vip.ogk.yahoo.co.jp/");  
  
        out.printf("\"%s\".equals(\"%s\") = %b\n", url1, url2, url1.equals(url2));  
    } catch (MalformedURLException e) { e.printStackTrace(); }  
}
```



<terminated> URLTest [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 11:20:26)
"http://yahoo.co.jp/".equals("http://f11.top.vip.ogk.yahoo.co.jp/") = true

参考：ドメイン名 \Leftrightarrow IPアドレス

- ❖ host コマンドを使用 (Windows では nslookup)



```
wakita@mini2013kw: ~ — ~ — zsh — 80x24
➔ ~ host yahoo.co.jp
yahoo.co.jp has address 124.83.187.140
yahoo.co.jp has address 203.216.243.240
yahoo.co.jp mail is handled by 10 mx3.mail.yahoo.co.jp.
yahoo.co.jp mail is handled by 10 mx5.mail.yahoo.co.jp.
yahoo.co.jp mail is handled by 10 mx1.mail.yahoo.co.jp.
yahoo.co.jp mail is handled by 10 mx2.mail.yahoo.co.jp.
➔ ~
➔ ~ host 124.83.187.140
140.187.83.124.in-addr.arpa domain name pointer f11.top.vip.ogk.yahoo.co.jp.
➔ ~
➔ ~ host f11.top.vip.ogk.yahoo.co.jp
f11.top.vip.ogk.yahoo.co.jp has address 124.83.187.140
➔ ~
```

equals の 雛形

```
public class Template {  
    // 値として重要なものを表現するフィールド  
    int x, y;  
    long[] v;  
    List<Object> list;  
    // 値としては重要でないフィールド  
    int a, b, c;  
  
    public boolean equals(Object o) {  
        // 比較のための計算が大変な場合には、this との == テストが高速化に有効  
        if (o == this)  
            return true;  
        // 引数に与えられたオブジェクトがこのクラスのインスタンスであることを確認  
        if (o instanceof Template) {  
            // このクラスに型変換  
            Template t = (Template) o;  
            if (!(t.x == x && t.y == y)) return false;  
            if (!java.util.Arrays.equals(v, t.v)) return false;  
            List<Object> tv = t.list;  
            if (!(tv.size() == list.size())) return false;  
            for (int i = 0; i < list.size(); i++)  
                if (!tv.get(i).equals(list.get(i))) return false;  
            return true;  
        }  
        // 引数が null の場合は instanceof は false なので以下になる。  
        return false;  
    }  
}
```


IX: *Always* override hashCode
when you override equals

equals を上書き \Rightarrow hashCode も

hashCode は簡約同値関係

ハッシュ法の動機

- ✧ ある全体集団の部分集合を効率的に表したい

集合の表現法 (1)

- ❖ 全体集合がかなり小さい(数十)場合 → EnumSet (～ビットベクトル)
- ❖ Enum Color { BLACK, WHITE, RED, ... }
EnumSet<Color> colors =
EnumSet.of(Color.RED, Color.WHITE);

集合の表現法 (2)

- ❖ 全体集合の索引（データに対する番号付け）が密で、
表したい集合と全体集合の大きさがあまり変わらない
場合 → 配列

- ❖ 例: シリアル番号の集合

```
boolean[] serial;
```

```
boolean isMember = serial(10356);
```

集合の表現法 (3)

- ❖ 全体集合がとてつもなく大きいが、表現したい集合がかなり小さい場合 → 線形リスト
- ❖

```
List<String> list = new ArrayList<String>();  
list.add("のび太"); list.add("ドラえもん");  
list.add("ジャイアン"); list.add("しずか");
```


集合の表現法 (4)

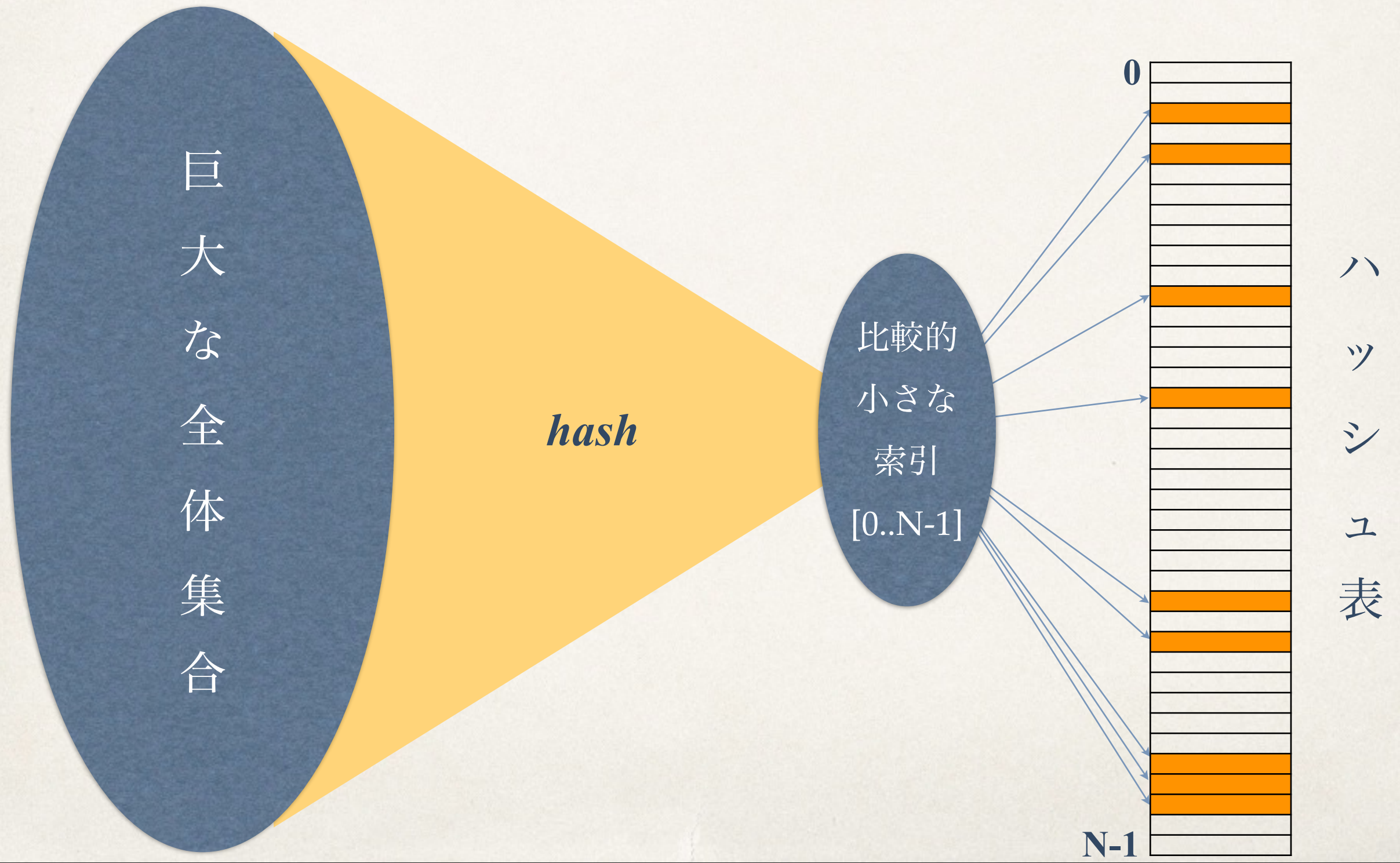
- ❖ 全体集合がとてつもなく大きく、表現したい集合の要素に全順序関係が定義されている場合 → 整序木 (Sorted Tree)
- ❖ **SortedSet**<String> set = new TreeSet<String>();
set.add("のび太"); set.add("ドラえもん");
set.add("ジャイアン"); set.add("しずか");
- ❖ 整序木の実装には**平衡二分探索木**を用いる。平衡二分探索木にはヒープ木の他に二色木、AVL木がある。平衡多分木としてはB木が有名。
 - ❖ 注：AVL木は平衡二分探索木だが、**平衡二分木**ではないことに注意。
- ❖ 集合の要素が変化しない場合は、配列に保存したものを最初にソートし、二分探索した方が高速。この場合、java.util.Arrays.{sort, binarysearch} が有用。

集合の表現法 (5)

- ❖ 全体集合がとてつもなく大きく，部分集合もかなり大きい場合 → ハッシュ

事例	全体集合	部分集合
全学生のID 10B0123-4	$102 \cdot 26 \cdot 105$ $= 260,000,000$	$4,660 + 118 + 1,524$ $= 6,302$ 人
Java API のクラス名 java.lang.Object	∞	3,793個
国語事典の見出し語	∞	26万語

ハッシュ法の概要



hashに求められる性質

- ❖ オブジェクトの値が変化しないうちは同じ値を返すこと
- ❖ $x.equals(y) \Rightarrow hash(x) == hash(y)$
- ❖ $!x.equals(y)$ なら高い率で
 $hash(x) != hash(y)$ となつて欲しい
- ❖ $hash(x) = (x.hashCode() \% \text{表の大きさ})$

hashCode を上書き
し忘れると？

```
class PhoneNumber {  
    private final short areaCode, prefix, number;  
  
    public PhoneNumber(int a, int p, int n) {  
        areaCode = rangeCheck(a, 999, "市街局番");  
        prefix = rangeCheck(p, 9999, "市内局番");  
        number = rangeCheck(n, 9999, "加入者番号");  
    }  
  
    private short rangeCheck(int arg, int max, String name) {  
        if (arg < 0 || arg > max)  
            throw new IllegalArgumentException(name + ": " + arg);  
        return (short)arg;  
    }  
  
    public boolean equals(Object o) {  
        if (o == this) return true;  
        if (!(o instanceof PhoneNumber)) return false;  
        PhoneNumber x = (PhoneNumber)o;  
        return x.areaCode == areaCode && x.prefix == prefix &&  
            x.number == number;  
    }  
}
```


テスト

```
private void run() {  
    PhoneNumber phone1 = new PhoneNumber(03, 5734, 3493);  
    PhoneNumber phone2 = new PhoneNumber(03, 5734, 3493);  
    out.printf("%sと%sの値は一致していま%s。 \n\n", phone1, phone2,  
        phone1.equals(phone2) ? "す" : "せん");  
  
    Set<PhoneNumber> 電話帳 = hashSet();  
  
    out.println(phone1 + "を追加します。");  
    電話帳.add(phone1);  
    out.println(java.util.Arrays.toString(電話帳.toArray()));  
  
    out.printf("\nお探しの電話番号(%s)が登録されてま%s。 \n\n", phone1,  
        電話帳.contains(phone2) ? "す" : "せん");  
  
    out.println(phone2 + "を追加します。");  
    電話帳.add(phone2);  
    out.println(java.util.Arrays.toString(電話帳.toArray()));  
}
```

テスト

Console

<terminated> HashTest1 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/ja

(03)5734-3493と(03)5734-3493の値は一致しています。

(03)5734-3493を追加します。

[(03)5734-3493]

お探しの電話番号((03)5734-3493)が登録されてません。

(03)5734-3493を追加します。

[(03)5734-3493, (03)5734-3493]

諸悪の根源:

phone1.hashCode() = 1867546546

phone2.hashCode() = 1298264335

hashCodeを上書き

```
public int hashCode() {  
    int p0 = 21, p = 31;  
    return ((p0 + areaCode) * p + prefix) * p + number;  
}
```

❖ p0, p: 素数を選ぶこと

❖ p = 31 だと乗算が効率的

$$x * p \Rightarrow x \ll 5 - x$$

テスト



```
Problems @ Javadoc Declaration Console Search
<terminated> HashTest2 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/ja
(03)5734-3493と(03)5734-3493の値は一致しています。

(03)5734-3493を追加します。
[(03)5734-3493]

お探しの電話番号((03)5734-3493)が登録されてます。

(03)5734-3493を追加します。
[(03)5734-3493]
```


hashCodeの設計上の注意

- ❖ equals の計算に用いないフィールドをhashCodeの計算に用いないこと \Rightarrow
$$x.equals(y) \Rightarrow x.hashCode() = y.hashCode()$$
- ❖ equals に影響のあるフィールドはもれなくhashCodeの計算に用いるのが吉

課題（ハッシュ）

- ❖ HashTest2 で定義したハッシュ関数を単純化して作成した HashTest3 もちゃんと動作しているように見える。アルゴリズムの教科書でハッシュ法を復習し、HashTest3 の性能上の問題点を挙げなさい。
- ❖ 上述の問題点が顕在化するような例題を作成し、詳細に調査しなさい。
- ❖ 以上を今週中にやること。ただし提出は不要。

まとめ

- ❖ equals: value class, equality(同値関係), 継承との相性の悪さ
- ❖ hashCode: equals と密接に関連