

# 計算機科学第一

2012年度第6回

# 理学部長を囲む昼食会

- 理学部長との昼食（無料）
- 11/28 (wed), 11/30 (fri): 12:20-13:00
- 本館1F146: 理学系長会議室
- 各回8名
- 西森秀稔より

授業や大学のことから全般について皆さんの率直なご意見を聞く機会を持ちます。ぜひお集まり下さい。

# ICPC Asia@Tokyo

- A: Ginkgo Numbers
- 実は素数の概念の自然数から複素数への拡張。
- 複素数に対する素数判定を行うプログラムを作成しなさい

# 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 <a href="#">Object</a>	<a href="#">clone()</a> このオブジェクトのコピーを作成し	<a href="#">String</a>	<a href="#">toString()</a> オブジェクトの文字列表現を返
boolean	<a href="#">equals(Object obj)</a> このオブジェクトと「等価」になる	void	<a href="#">wait()</a> ほかのスレッドがこのオブジェ
protected void	<a href="#">finalize()</a> このオブジェクトへの参照はもうな		させます。
<a href="#">Class</a> <?>	<a href="#">getClass()</a> この Object の実行時クラスを返し	void	<a href="#">wait(long timeout)</a> 別のスレッドがこのオブジェ
int	<a href="#">hashCode()</a> オブジェクトのハッシュコード値を		まで、現在のスレッドを待機させます
void	<a href="#">notify()</a> 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	<a href="#">wait(long timeout, int nanos)</a> ほかのスレッドがこのオブジェ
void	<a href="#">notifyAll()</a> 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。		スレッドに割り込みをかけたり、指定
<a href="#">String</a>	<a href="#">toString()</a> 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
equals	等価性（＝同値性）の判定
hashCode	ハッシュコードの計算
clone	コピーを作成
toString	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≠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:

$$x.equals(y) \wedge y.equals(z) \Rightarrow x.equals(z)$$

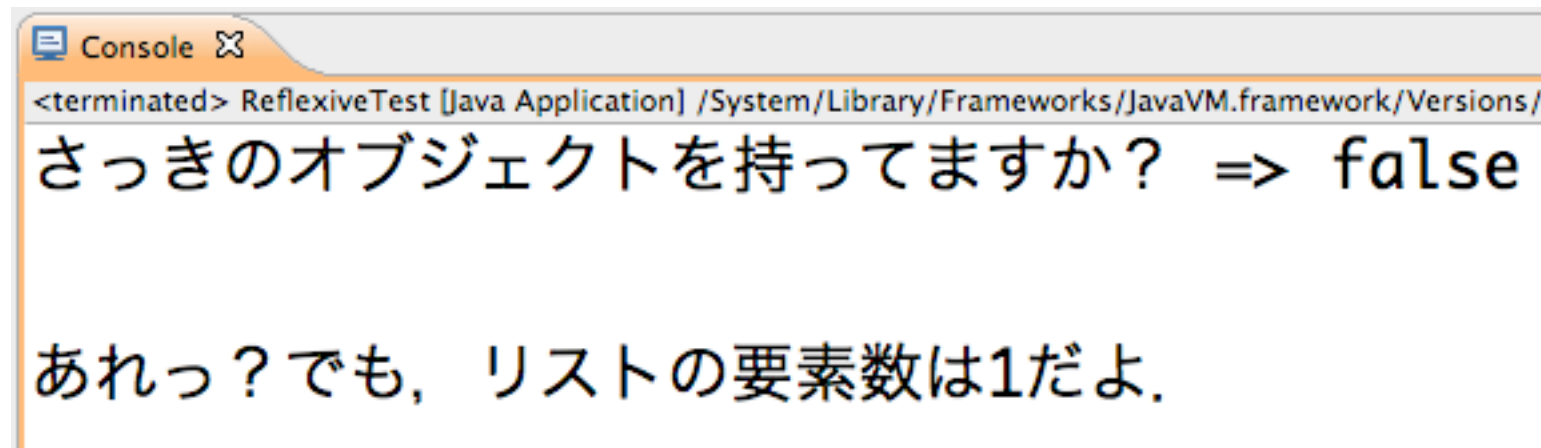
# 反射律を壊してみよう

```
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...



A screenshot of a Java console window. The title bar says "Console" with a close button. The text inside the console shows a terminated Java application named "ReflexiveTest" running on a specific Java VM. The output of the application is a Japanese question followed by an equals sign and the word "false". Below this, there is a Japanese comment.

```
<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.
```



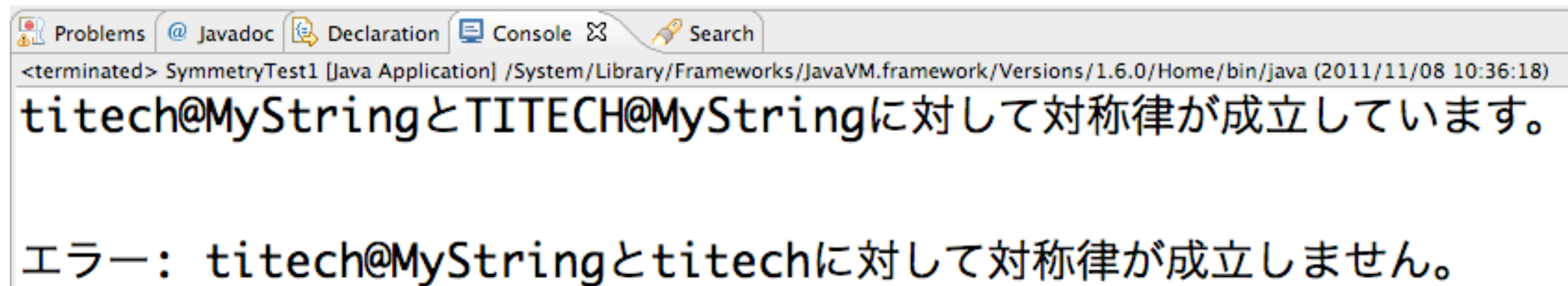
# 対称律を壊してみよう

```
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 a toolbar at the top containing icons for Problems, Javadoc, Declaration, Console, and Search. The console output is as follows:

```
<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

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

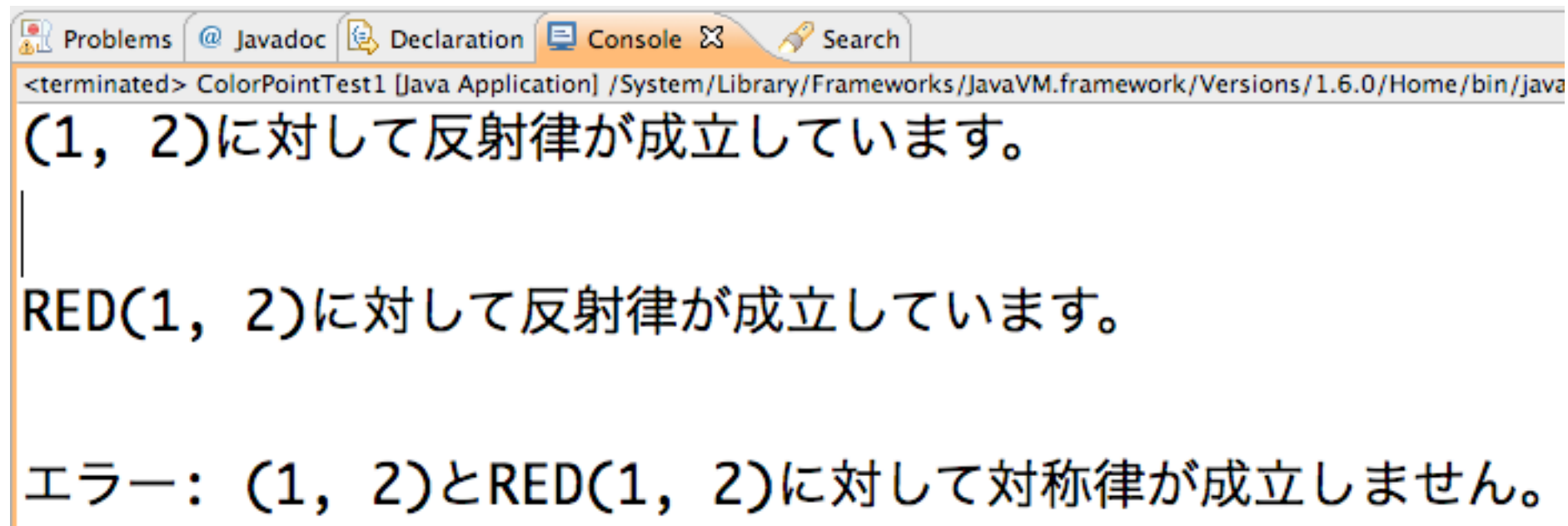
$$(x_p, y_p) = (x_q, y_q)$$

```
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

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

# ColorPointTest I



The screenshot shows a Java IDE console window with the following text:

```
<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

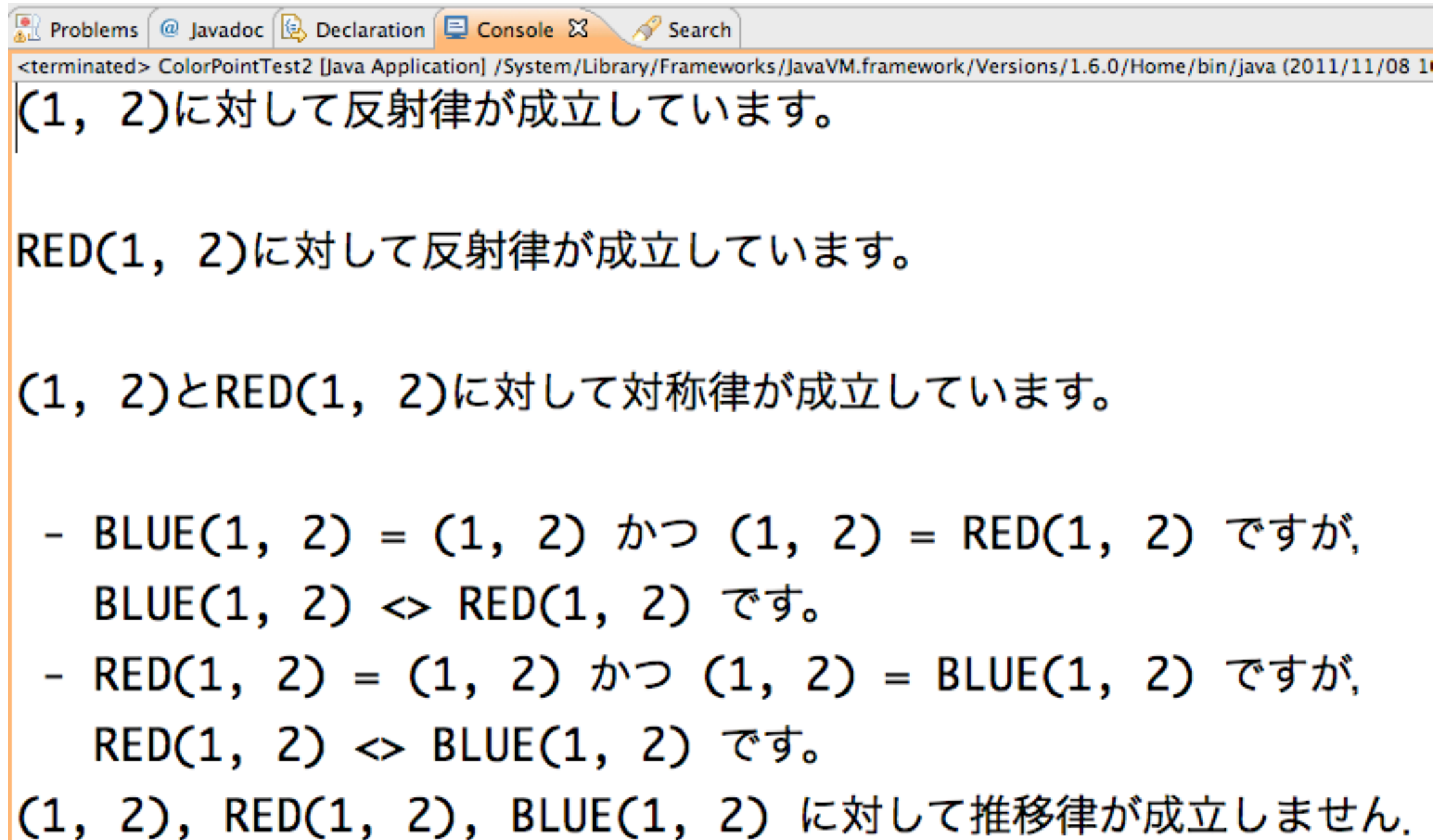
```
class Point {  
    protected final int x, y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public boolean equals(Object o) {  
        if (!(o instanceof Point)) return false;  
        Point p = (Point) o;  
        return p.x == x && p.y ==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

```
<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 1

(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)  $\neq$  RED(1, 2) です。
- RED(1, 2) = (1, 2) かつ (1, 2) = BLUE(1, 2) ですが, RED(1, 2)  $\neq$  BLUE(1, 2) です。

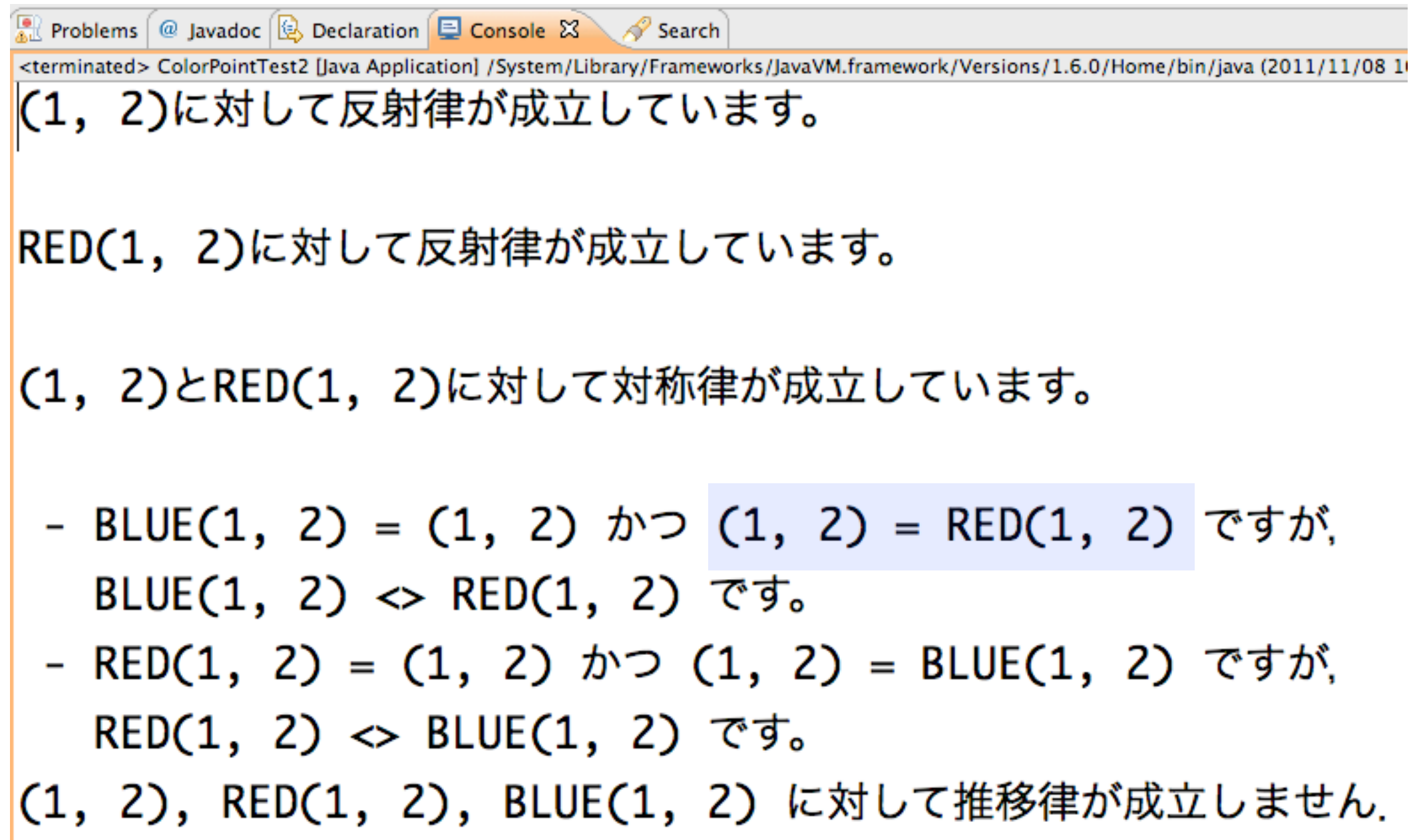
(1, 2), RED(1, 2), BLUE(1, 2) に対して推移律が成立しません。

クラスが異なるので、(1, 2).equals( RED(1, 2) ) を呼ぶ。

これは Color.equals → true



# テスト



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

```
<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。

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

(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 Search  
<terminated> ColorPointTest2 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 1

(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、同値性契約を満すのは困難

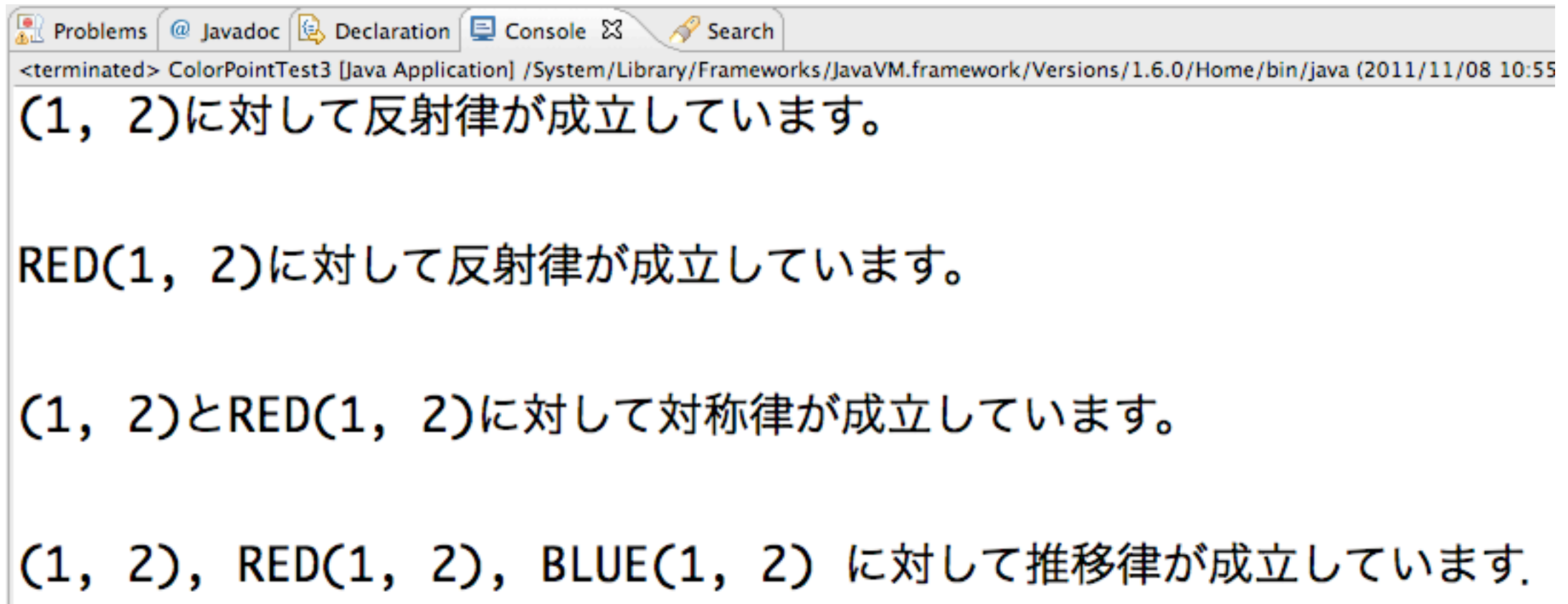
# クラス継承以外の拡張法

# 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 content:

```
<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。

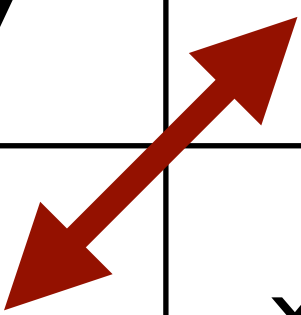
(1, 2)とRED(1, 2)に対して対称律が成立しています。

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

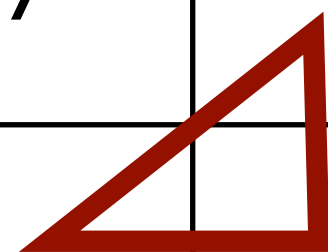
The console window has a toolbar with icons for Problems, Javadoc, Declaration, Console, and Search. The text in the console is black on a white background.



Test <sub>1</sub>	P <sub>2</sub>	CP <sub>2</sub>
P <sub>1</sub>	x, y	x, y
CP <sub>1</sub>	F	x, y, c



Test <sub>2</sub>	P <sub>2</sub>	CP <sub>2</sub>
P <sub>1</sub>	x, y	x, y
CP <sub>1</sub>	x, y	x, y, c



Test <sub>3</sub>	P <sub>2</sub>	CP <sub>2</sub>
P <sub>1</sub>	x, y	F
CP <sub>1</sub>	F	p, c



# 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。

```
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。

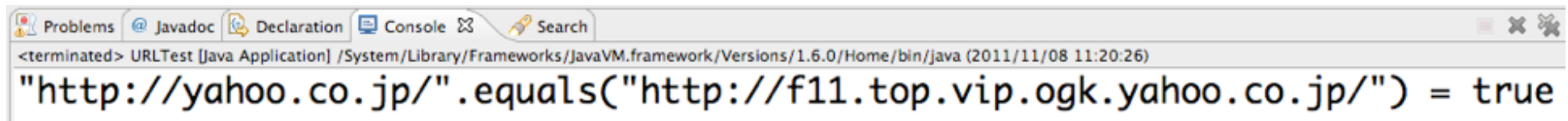
2つのホストが等価と見なされるのは、両方のホスト名が同じ IP アドレスに解決されるか、どちらかのホスト名を解決できない場合は、大文字小文字に関係なくホスト名が等しいか、両方のホスト名が `null` に等しい場合です。

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(); }  
}
```



The screenshot shows an IDE interface with a console window. The console title bar includes tabs for Problems, Javadoc, Declaration, Console, and Search. The console output shows the application has terminated and displays the result of the equals method call between two URLs.

```
<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
```

```

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 ⇒  
`hashCode` も上書き

`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 (I)

- 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）構築が容易で、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 Vector<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)

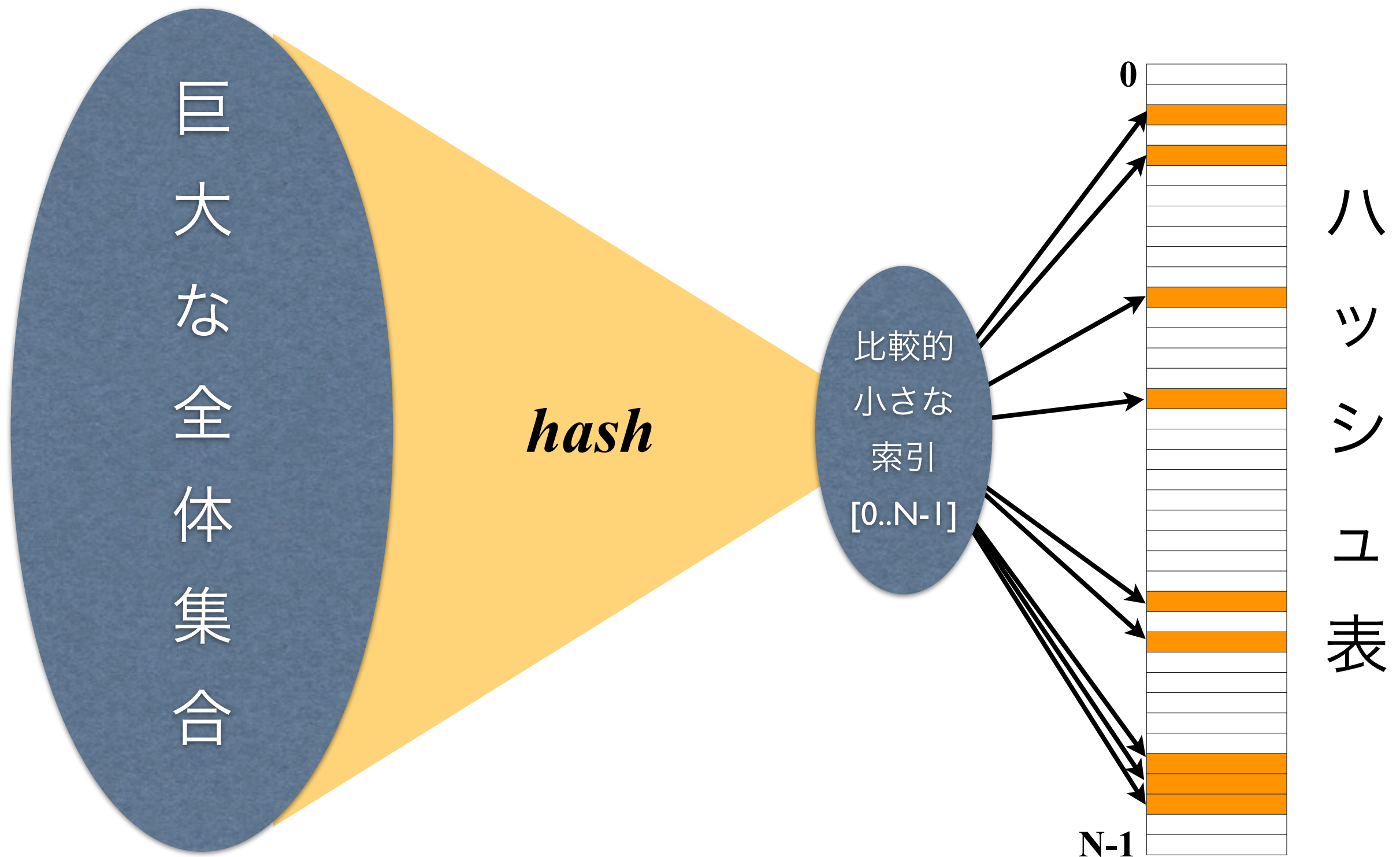
- 全体集合がとてつもなく大きい、表現したい集合の要素に全順序関係が定義されている場合 → ヒープ木※
- `Set<String> set = new TreeSet<String>();`  
`set.add(“のび太”); set.add(“ドラえもん”);`  
`set.add(“ジャイアン”); set.add(“しずか”);`
- ヒープ木※: 平衡二分木の一つ

# 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	$10^2 \cdot 26 \cdot 10^4$ = 26,000,000	$4,660 + 118 + 1,524$ = 6,302人
Java API のクラス名 java.lang.Object	$\infty$	3,793個
国語事典の見出し語	$\infty$	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;
    }
}

```

# テストコード

```
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す。 \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
```

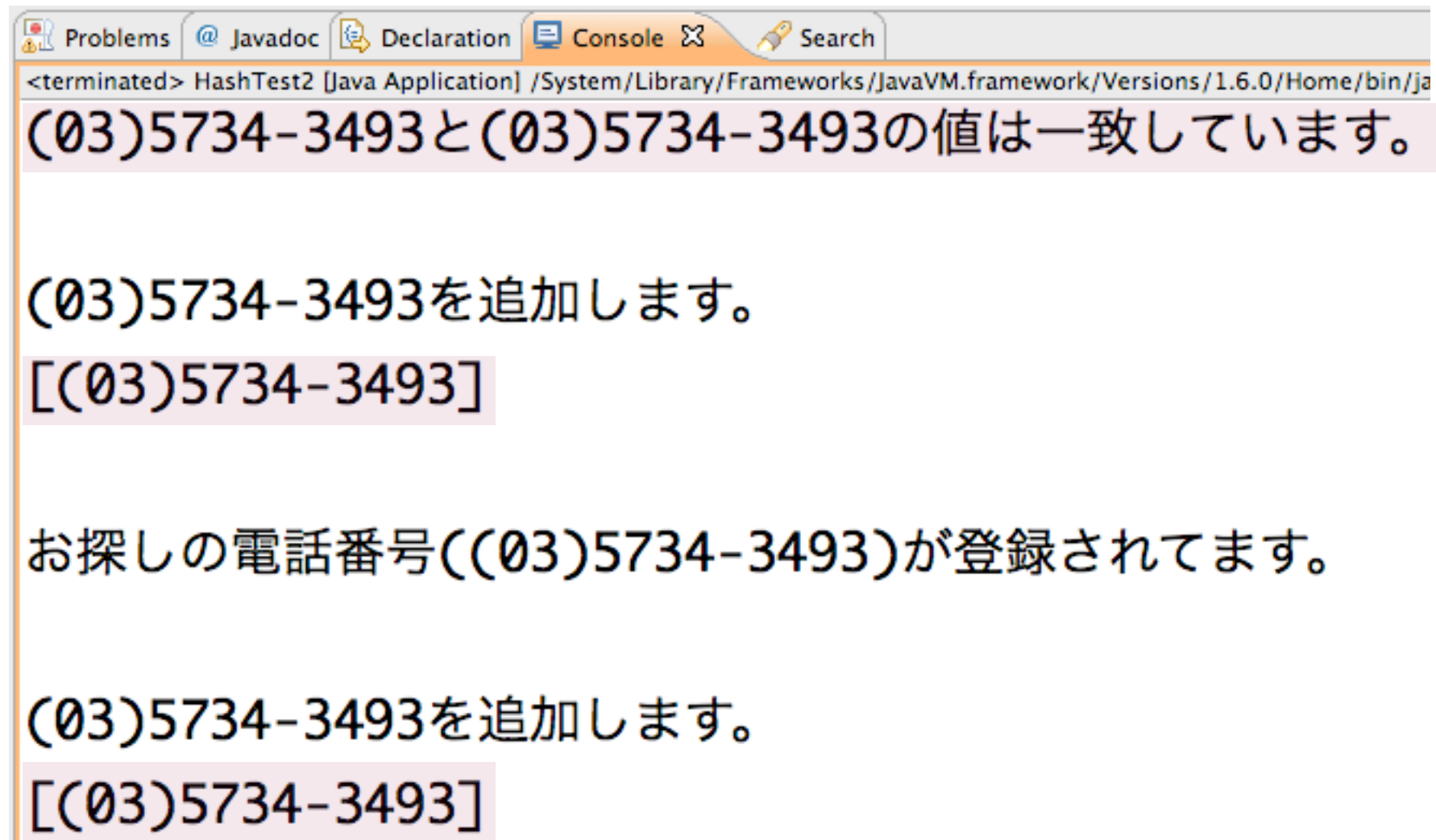
# hashCodeを上書き

```
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

$$x * p \Rightarrow x \ll 5 + 1$$

# テスト



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

```
<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]
```

The console window has tabs for Problems, Javadoc, Declaration, Console, and Search. The Console tab is active, showing the output of the HashTest2 application. The output consists of several lines of text, including a confirmation message, two addition prompts, and two array representations of the phone number.

# 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

# 宿題

- 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。

***X: Always override  
toString***



普通 toString は  
上書きするでしょ

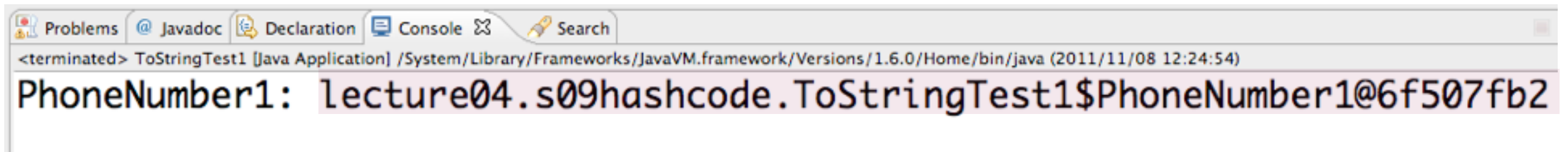
# toString を上書きしない場合

```
class PhoneNumber1 {  
    private final short areaCode, prefix, number;  
  
    public PhoneNumber1(int a, int p, int n) {  
        areaCode = (short)a;  
        prefix = (short)p;  
        number = (short)n;  
    }  
}
```

# 普通に出力してみる

```
private void print(Object o) {  
    out.printf("%s: %s\n\n", o.getClass().getSimpleName(), o);  
}
```

```
private void run() {  
    print(new PhoneNumber1(03, 5734, 3493));  
}
```



The screenshot shows an IDE interface with a 'Console' tab selected. The console output displays the result of a Java application run, showing the class name and the phone number data.

```
<terminated> ToStringTest1 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 12:24:54)  
PhoneNumber1: lecture04.s09hashCode.ToStringTest1$PhoneNumber1@6f507fb2
```

# toString を上書きすると

```
public String toString() {  
    return String.format("(%02d)%04d-%04d", areaCode, prefix, number);  
}
```



The screenshot shows a Java IDE console window with the following content:

```
<terminated> ToStringTest1 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 12:24:54)  
PhoneNumber1: lecture04.s09hashCode.ToStringTest1$PhoneNumber1@6f507fb2  
  
PhoneNumber2: (03)5734-3493
```

The output for PhoneNumber2 is highlighted with a light pink background.

# toString を上書きすると

```
private void run() {  
    PhoneNumber1 phone1 = new PhoneNumber1(03, 5734, 3493);  
    PhoneNumber2 phone2 = new PhoneNumber2(03, 5734, 3493);  
    print(phone1);  
    print(phone2);  
    out.printf("phone1.hashCode = %x\n", phone1.hashCode());  
}
```



The screenshot shows a Java IDE console window with the following output:

```
<terminated> ToStringTest1 [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (2011/11/08 12:30:43)  
PhoneNumber1: lecture04.s09hashcode.ToStringTest1$PhoneNumber1@6f507fb2  
  
PhoneNumber2: (03)5734-3493  
  
phone1.hashCode = 6f507fb2
```

# toString を上書きしたら やっておきたいこと

- toString の出力形式を使った  
Constructor
- toString に関わるフィールドへのアクセ  
サ

# 文字列 → PhoneNumber

```
public static PhoneNumber of(String s) {  
    Scanner scan = new Scanner(s);  
    scan.next("\\([([0-9]+)\\)([0-9]+)-([0-9]+)");  
    MatchResult match = scan.match();  
    return new PhoneNumber(Short.valueOf(match.group(1)),  
                           Short.valueOf(match.group(2)),  
                           Short.valueOf(match.group(3)));  
}
```

Java でも正規表現(Regular expression) が  
使えます。

# PhoneNumber のアクセス

```
public short areaCode() { return areaCode; }  
public short prefix() { return prefix; }  
public short number() { return number; }
```



# 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
- toString: 見易い出力のために