

計算機科学第一

2012年度第5回

もくじ

- Item 3: Singleton
- Item 4: Private constructor
- Item 5: Avoid creating

4. Private Constructor

“new”させねえ

“new”させたくない場合

- static method だけからなるクラス
*Utility*クラス

Utilityの例: java.lang.Math

フィールドの概要

static double	E 自然対数の底 e にもっとも近い <i>double</i> 値です。
static double	PI 円周とその直径の比 π にもっとも近い <i>double</i> 値です。

メソッドの概要

static double	abs (double a) double 値の絶対値を返します。
static float	abs (float a) float 値の絶対値を返します。
static int	abs (int a) int 値の絶対値を返します。
static long	abs (long a) long 値の絶対値を返します。
static double	acos (double a) 指定された値の逆余弦 (アークコサイン) を返します。
static double	asin (double a) 指定された値の逆正弦 (アークサイン) を返します。
static double	atan (double a) 指定された値の逆正接 (アークタンジェント) を返します。
static double	atan2 (double y, double x) 極座標 (r, <i>theta</i>) への矩形座標 (x, y) の変換から角度 <i>theta</i> を返します。
static double	cbrt (double a) double 値の立方根を返します。

Utilityの例: java.util.Arrays

メソッドの概要

<code>static <T> List<T></code>	<code>asList(T... a)</code> 指定された配列に連動する固定サイズのリストを返します。
<code>static int</code>	<code>binarySearch(byte[] a, byte key)</code> バイナリサーチアルゴリズムを使用して、指定された byte 値の配列から指定された値を検
<code>static int</code>	<code>binarySearch(byte[] a, int fromIndex, int toIndex, byte key)</code> バイナリサーチアルゴリズムを使用して、指定された byte 値の配列から指定された値の範 す。
<code>static int</code>	<code>binarySearch(char[] a, char key)</code> バイナリサーチアルゴリズムを使用して、指定された char 値の配列から指定された値を検
<code>static int</code>	<code>binarySearch(char[] a, int fromIndex, int toIndex, char key)</code> バイナリサーチアルゴリズムを使用して、指定された char 値の配列から指定された値の範 す。
<code>static int</code>	<code>binarySearch(double[] a, double key)</code> バイナリサーチアルゴリズムを使用して、指定された double 値の配列から指定された値を

Utilityの例: java.util.Collections

メソッドの概要	
<code>static <T> boolean</code>	<code>addAll(Collection<? super T> c, T... elements)</code> 指定されたすべての要素を指定されたコレクションに追加します。
<code>static <T> Queue<T></code>	<code>asLifoQueue(Deque<T> deque)</code> <code>Deque</code> のビューを後入れ先出し (Lifo) <code>Queue</code> として返します。
<code>static <T> int</code>	<code>binarySearch(List<? extends Comparable<? super T>> list, T key)</code> バイナリサーチアルゴリズムを使用して、指定されたリストから指定されたオブジェクトを検索します。
<code>static <T> int</code>	<code>binarySearch(List<? extends T> list, T key, Comparator<? super T> c)</code> バイナリサーチアルゴリズムを使用して、指定されたリストから指定されたオブジェクトを検索します。
<code>static <E> Collection<E></code>	<code>checkedCollection(Collection<E> c, Class<E> type)</code> 指定されたコレクションの、動的に型保証されたビューを返します。
<code>static <E> List<E></code>	<code>checkedList(List<E> list, Class<E> type)</code> 指定されたリストの動的に型保証されたビューを返します。
<code>static <K,V> Map<K,V></code>	<code>checkedMap(Map<K,V> m, Class<K> keyType, Class<V> valueType)</code> 指定されたマップの動的に型保証されたビューを返します。
<code>static <E> Set<E></code>	<code>checkedSet(Set<E> s, Class<E> type)</code> 指定されたセットの動的に型保証されたビューを返します。
<code>static <K,V> SortedMap<K,V></code>	<code>checkedSortedMap(SortedMap<K,V> m, Class<K> keyType, Class<V> valueType)</code> 指定されたソートマップの動的に型保証されたビューを返します。
<code>static <E> SortedSet<E></code>	<code>checkedSortedSet(SortedSet<E> s, Class<E> type)</code> 指定されたソートセットの動的に型保証されたビューを返します。

Utilityクラスは“new” させたくない



100644 | 16 lines (13 sloc) | 0.528 kb

```
1  package lecture02.s4private_constructor;
2
3  import java.util.HashMap;
4  import java.util.List;
5  import java.util.Map;
6  import java.util.TreeMap;
7  import java.util.Vector;
8
9  public class Tools {
10     public static <V> List<V> vector() { return new Vector<V>(); }
11     public static <K, V> Map<K, V> hashMap() { return new HashMap<K, V>(); }
12     public static <K, V> Map<K, V> treeMap() { return new TreeMap<K, V>(); }
13     // 以下の宣言で Tools クラスのインスタンスの生成を禁止している。
14     private Tools() {}
15 }
16
```


Utilityクラスは“new” させたくない



100644 | 16 lines (13 sloc) | 0.528 kb

```
1 package lecture02.s4private_constructor;
2
3 import java.util.HashMap;
4 import java.util.List;
5 import java.util.Map;
6 import java.util.TreeMap;
7 import java.util.Vector;
8
9 public class Tools {
10     public static <V> List<V> vector() { return new Vector<V>(); }
11     public static <K, V> Map<K, V> hashMap() { return new HashMap<K, V>(); }
12     public static <K, V> Map<K, V> treeMap() { return new TreeMap<K, V>(); }
13     // 以下の宣言で Tools クラスのインスタンスの生成を禁止している。
14     private Tools() {}
15 }
16
```

constructor を private にするとどうなる？

まとめ: private constructor

- 技術的には簡単で、応用範囲が広い
- Utility クラスの有用性を学んで欲しい

3. Singleton

この世の中で唯一無二のあなた

Singleton

- 「インスタンスをたった一つ持つことを宿命づけられたクラス」
- 語源: 集合論の singleton (単集合)
- システムに本質的に一つしか存在しないものを表す。
- 応用例: ウィンドウマネージャ, ファイルシステム, ODBC
- 存在意義: プログラムのなかで複数の並列実行単位 (thread) が, Singleton クラスをそれぞれ new し, 本来, 一つしかないものを表すオブジェクトが複数できると、データの一貫性保持や並行性制御が困難・複雑になる。

実装方法

- 伝統的な方法
 - static field + private constructor
 - static factory + private constructor
- 最新かつ簡単な方法（一押し）
 - Java 5 から導入された **Enum**

Singleton@static field

100644 | 13 lines (9 sloc) | 0.455 kb

```
1 package lecture02.s3singleton;
2
3 public class Singleton1 implements Singleton {
4     public static final Singleton INSTANCE = new Singleton1();
5
6     // Item4: Private constructor によってインスタンスの生成を禁止している。
7     private Singleton1() {}
8
9     // 以下はおまけ
10    private static final String name = "この世の中で唯一無二の存在";
11    public String toString() { return "私は" + name + "です。"; }
12 }
13
```

Singleton@static factory

```
100644 | 14 lines (10 sloc) | 0.532 kb

1  package lecture02.s3singleton;
2
3  public class Singleton2 implements Singleton {
4      // Item 1: static factory method を用いた実装
5      private static final Singleton2 INSTANCE = new Singleton2();
6      public static Singleton2 getInstance() { return INSTANCE; }
7
8      // Item 4: private constructor で new できないようにする。
9      private Singleton2() {}
10
11     private static final String name = "この世の中で唯一無二の存在";
12     public String toString() { return "私は" + name + "です。"; }
13 }
```

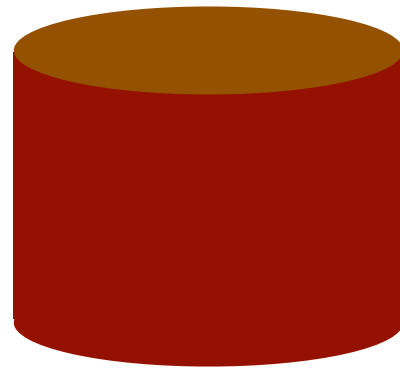
Java 5より古い時代のトレンドはこちら

Singletonの落とし穴

Java の世界

Singleton

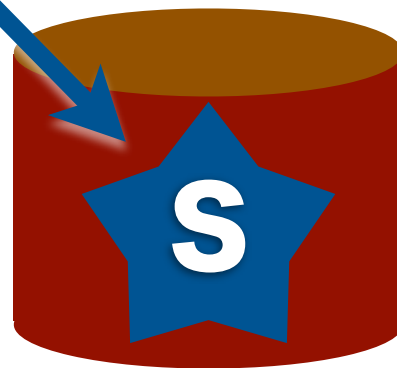
...



ハードディスク

Singletonの落とし穴

Java の世界



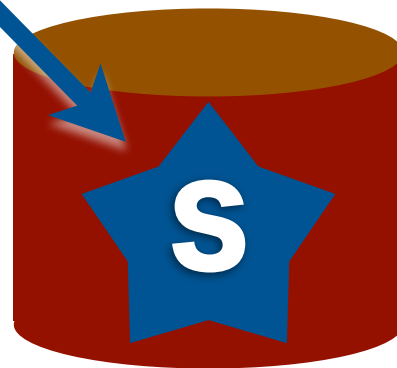
ハードディスク

Singletonの落とし穴

Java の世界



serialize
直列化ともいう



ハードディスク

Singletonの落とし穴

Java の世界

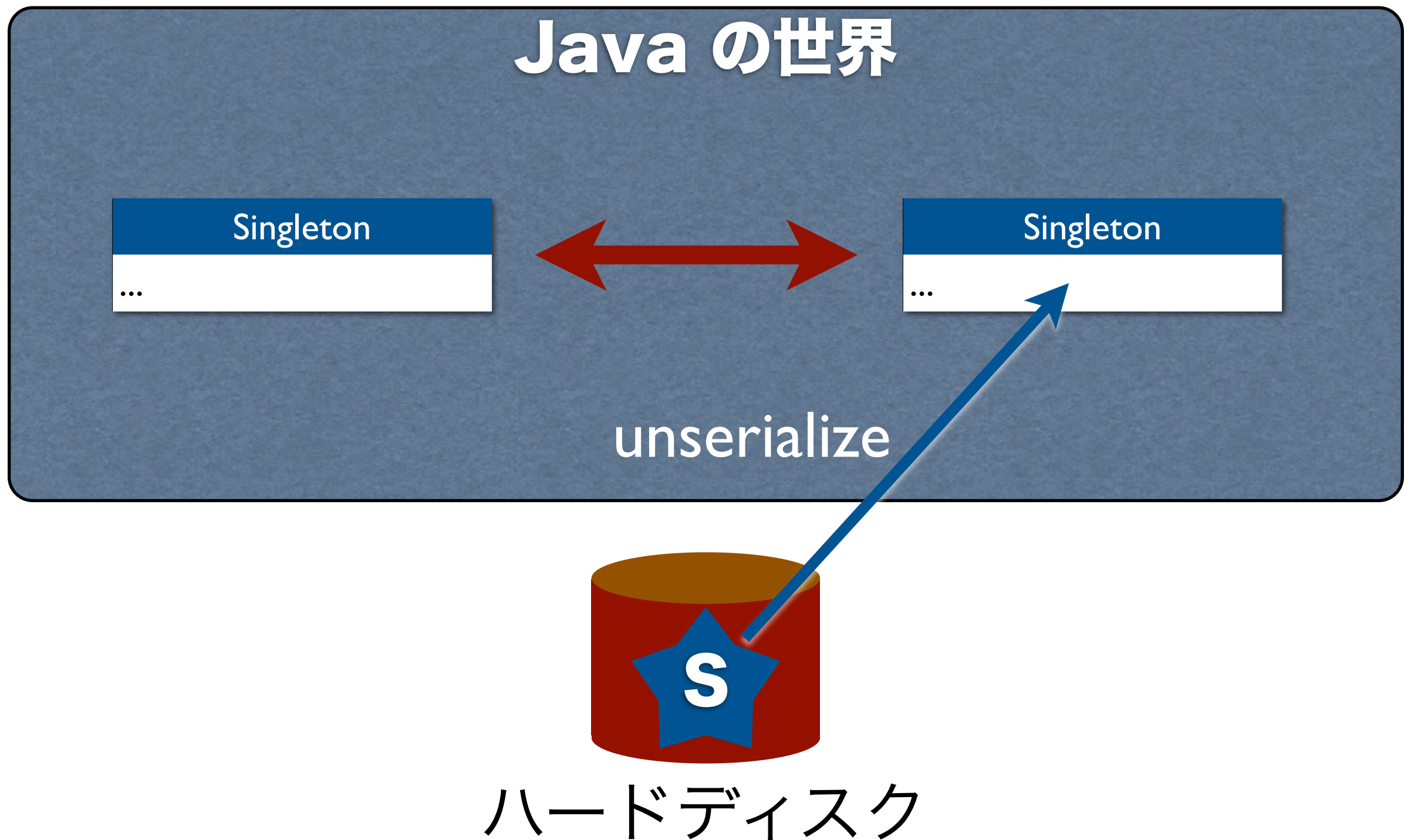


unserializeした
オブジェクトは？



ハードディスク

Singletonの落とし穴



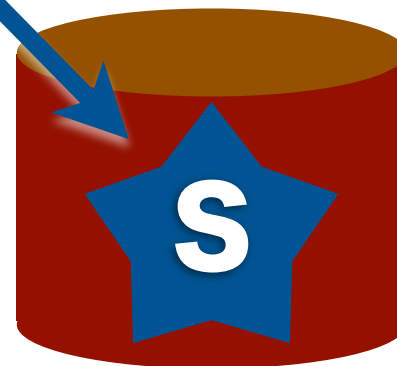
Singletonの落とし穴

Javaの世界



unserialize

readResolveメソッドを
カスタマイズして対応



Javaの実行環境の外の世界：

たとえば、ハードディスクやインターネット



100644 | 17 lines (12 sloc) | 0.665 kb

```
1 package lecture02.s3singleton;
2
3 import java.io.Serializable;
4
5 public class Singletonlb implements Singleton, Serializable {
6     private static final long serialVersionUID = -854105007273143543L;
7     // 複製を抑制するために、unserializeのときに呼ばれるメソッドを上書きし、
8     // 定数を返すことで複製の作成を抑制
9     private Object readResolve() { return INSTANCE; }
10
11     public static final Singleton INSTANCE = new Singletonlb();
12
13     private Singletonlb() {}
14     private static final String name = "この世の中で唯一無二の存在";
15     public String toString() { return "私は" + name + "です。"; }
16 }
```

Enum を使えばずっと簡単



100644 | 9 lines (6 sloc) | 0.252 kb

```
1 package lecture02.s3singleton;
2
3 public enum Singleton3 implements Singleton {
4     INSTANCE;
5
6     private static final String name = "この世の中で唯一無二の存在";
7     public String toString() { return "私は" + name + "です。"; }
8 }
```

Enum を使えばずっと簡単



100644 | 9 lines (6 sloc) | 0.252 kb

```
1 package lecture02.s3singleton;
2
3 public enum Singleton3 implements Singleton {
4     INSTANCE;
5
6     private static final String name = "この世の中で唯一無二の存在";
7     public String toString() { return "私は" + name + "です。"; }
8 }
```


Enum とは

- 複数の区別したい定数を列挙したもの
- 例：曜日、信号の三色、選択肢の候補、などなど

Enum を使えばずっと簡単

```
100644 | 9 lines (6 sloc) | 0.252 kb

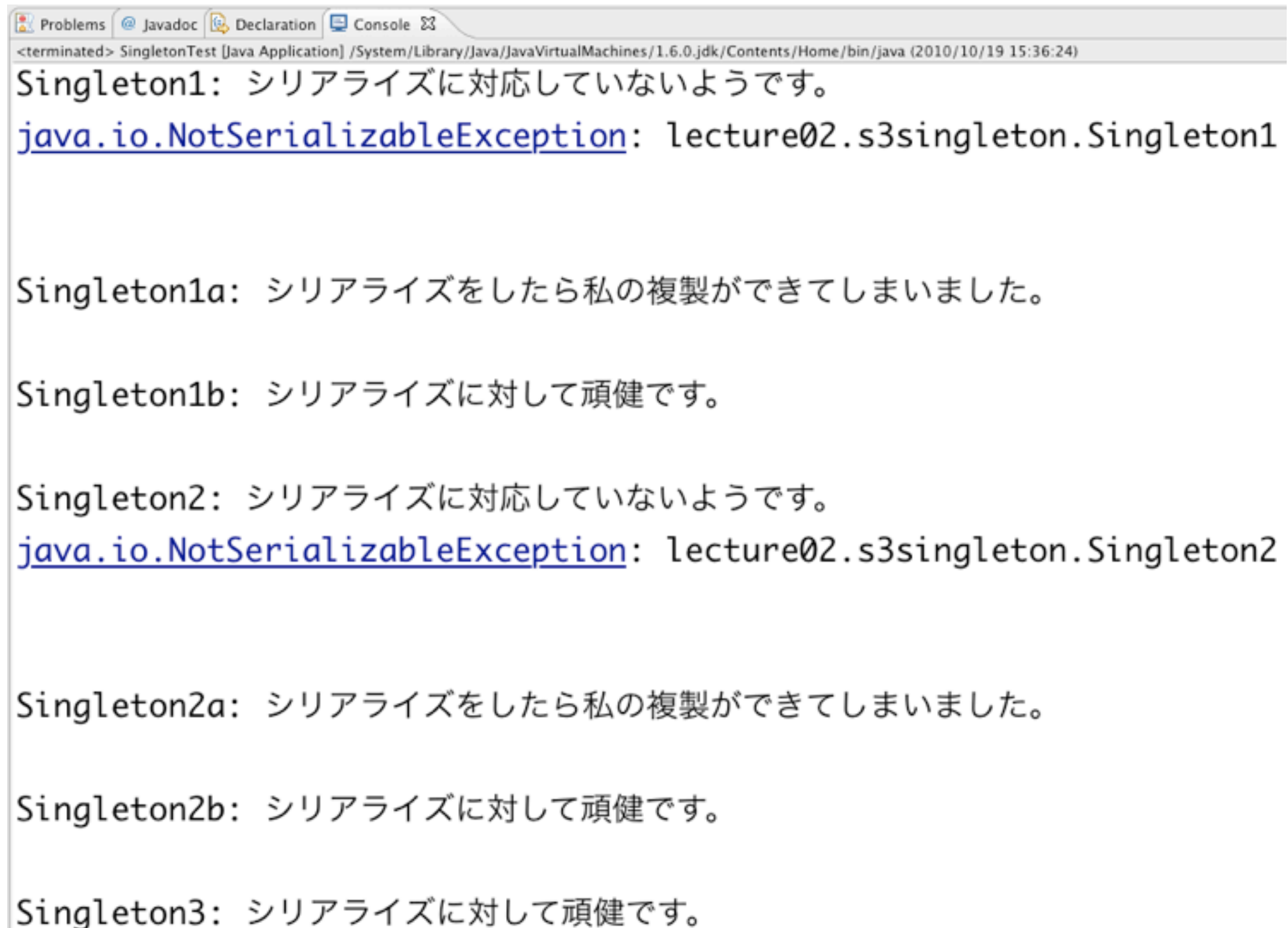
1 package lecture02.s3singleton;
2
3 public enum Singleton3 implements Singleton {
4     INSTANCE;
5
6     private static final String name = "この世の中で唯一無二の存在";
7     public String toString() { return "私は" + name + "です。"; }
8 }
```

そもそも new のない言語機構だから、
serialize/unserializeの問題もなし

実験: SingletonTest.java

```
9 public class SingletonTest {
10     private void test(String message, Singleton s1) {
11         System.out.printf("%s: %s\n", message, s1);
12         try {
13             Singleton s2 = (Singleton)unserialize(serialize(s1));
14             if (s1 != s2)
15                 System.err.println("シリアライズをしたら私の複製ができてしまいました。");
16             else {
17                 System.out.println("この実装はシリアライズに対して頑健です。");
18                 System.out.flush();
19             }
20         } catch (Exception e) {
21             System.err.printf("Serialize/Unserialize の途中で例外が発生しました。 \n%s", e);
22         }
23     }
24
25     private void run() {
26         test("Singleton1a", Singleton1a.INSTANCE);
27         test("Singleton1b", Singleton1b.INSTANCE);
28         test("Singleton2a", Singleton2a.getInstance());
29         test("Singleton2b", Singleton2b.getInstance());
30         test("Singleton3", Singleton3.INSTANCE);
31     }
```

実験結果: SingletonTest.java



```
<terminated> SingletonTest [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (2010/10/19 15:36:24)
Singleton1: シリアライズに対応していないようです。
java.io.NotSerializableException: lecture02.s3singleton.Singleton1

Singleton1a: シリアライズをしたら私の複製ができてしまいました。

Singleton1b: シリアライズに対して頑健です。

Singleton2: シリアライズに対応していないようです。
java.io.NotSerializableException: lecture02.s3singleton.Singleton2

Singleton2a: シリアライズをしたら私の複製ができてしまいました。

Singleton2b: シリアライズに対して頑健です。

Singleton3: シリアライズに対して頑健です。
```

まとめ: Singleton

- Singleton の実装方法としては、static field や static factory method を用いる方法が知られていたが、serialize 周りで苦労するので注意すること
- Java 5 で導入された Enum を用いるのが吉！

5. Avoid Creation

無駄に “new” しない

これだけはしないで

- `String s = new String("This is a pen");`

vs

- `String s = "This is a pen";`

これだけはしないで

- ~~String s = new String("This is a pen");~~

vs

- String s = "This is a pen";

次のプログラムの 問題点はなに？

ヒント：無駄を探せ

```
import java.util.Calendar;

public class Person1 implements Person {
    private final Date birthDate;

    public Person1(int year, int month, int date) {
        Calendar 日本標準時 = Calendar.getInstance(TimeZone.getTimeZone("JST"));
        日本標準時.set(year, month, date);
        birthDate = 日本標準時.getTime();
    }

    public boolean 昭和生れですか?() {
        Calendar 日本標準時 = Calendar.getInstance(TimeZone.getTimeZone("JST"));
        日本標準時.set(1926, Calendar.DECEMBER, 24); // 1926/12/25 0:0
        Date 昭和最初の日 = 日本標準時.getTime();
        日本標準時.set(1989, Calendar.JANUARY, 7); // 1989/1/8 0:0
        Date 昭和最後の日 = 日本標準時.getTime();
        return (birthDate.compareTo(昭和最初の日) >= 0 && birthDate
                .compareTo(昭和最後の日) < 0);
    }
}
```

改良版のプログラム

どこが変った？

効率は改善されたの？

```

public class Person2 implements Person {
    private final Date        birthDate;
    private static final Calendar 日本標準時 = Calendar.getInstance(TimeZone
                                                                    .getTimeZone("JST"));

    public Person2(int year, int month, int date) {
        日本標準時.set(year, month, date);
        birthDate = 日本標準時.getTime();
    }

    private static final Date 昭和最初の日, 昭和最後の日;
    static {
        日本標準時.set(1926, Calendar.DECEMBER, 24); // 1926/12/25 0:0
        昭和最初の日 = 日本標準時.getTime();

        日本標準時.set(1989, Calendar.JANUARY, 7); // 1989/1/8 0:0
        昭和最後の日 = 日本標準時.getTime();
    }

    public boolean 昭和生れですか?() {
        return (birthDate.compareTo(昭和最初の日) >= 0 && birthDate
                .compareTo(昭和最後の日) < 0);
    }
}

```

```

public class Person2 implements Person {
    private final Date        birthDate;

    private static final Calendar 日本標準時 = Calendar.getInstance(TimeZone
                                                                    .getTimeZone("JST"));

    public Person2(int year, int month, int date) {
        日本標準時.set(year, month, date);
        birthDate = 日本標準時.getTime();
    }

    private static final Date 昭和最初の日, 昭和最後の日;
    static {
        日本標準時.set(1926, Calendar.DECEMBER, 24); // 1926/12/25 0:0
        昭和最初の日 = 日本標準時.getTime();

        日本標準時.set(1989, Calendar.JANUARY, 7); // 1989/1/8 0:0
        昭和最後の日 = 日本標準時.getTime();
    }

    public boolean 昭和生れですか?() {
        return (birthDate.compareTo(昭和最初の日) >= 0 && birthDate
                .compareTo(昭和最後の日) < 0);
    }
}

```



```

public class Person2 implements Person {
    private final Date        birthDate;
    private static final Calendar 日本標準時 = Calendar.getInstance(TimeZone
                                                                    .getTimeZone("JST"));

    public Person2(int year, int month, int date) {
        日本標準時.set(year, month, date);
        birthDate = 日本標準時.getTime();
    }

    private static final Date 昭和最初の日, 昭和最後の日;
    static {
        日本標準時.set(1926, Calendar.DECEMBER, 24); // 1926/12/25 0:0
        昭和最初の日 = 日本標準時.getTime();

        日本標準時.set(1989, Calendar.JANUARY, 7); // 1989/1/8 0:0
        昭和最後の日 = 日本標準時.getTime();
    }

    public boolean 昭和生れですか?() {
        return (birthDate.compareTo(昭和最初の日) >= 0 && birthDate
                .compareTo(昭和最後の日) < 0);
    }
}

```

```

public class Person2 implements Person {
    private final Date        birthDate;
    private static final Calendar 日本標準時 = Calendar.getInstance(TimeZone
                                                                    .getTimeZone("JST"));

    public Person2(int year, int month, int date) {
        日本標準時.set(year, month, date);
        birthDate = 日本標準時.getTime();
    }

    private static final Date 昭和最初の日, 昭和最後の日;
    static {
        日本標準時.set(1926, Calendar.DECEMBER, 24); // 1926/12/25 0:0
        昭和最初の日 = 日本標準時.getTime();

        日本標準時.set(1989, Calendar.JANUARY, 7); // 1989/1/8 0:0
        昭和最後の日 = 日本標準時.getTime();
    }

    public boolean 昭和生れですか?() {
        return (birthDate.compareTo(昭和最初の日) >= 0 && birthDate
                .compareTo(昭和最後の日) < 0);
    }
}

```

```

public class Person2 implements Person {
    private final Date        birthDate;
    private static final Calendar 日本標準時 = Calendar.getInstance(TimeZone
                                                                    .getTimeZone("JST"));

    public Person2(int year, int month, int date) {
        日本標準時.set(year, month, date);
        birthDate = 日本標準時.getTime();
    }

    private static final Date 昭和最初の日, 昭和最後の日;
    static {
        日本標準時.set(1926, Calendar.DECEMBER, 24); // 1926/12/25 0:0
        昭和最初の日 = 日本標準時.getTime();

        日本標準時.set(1989, Calendar.JANUARY, 7); // 1989/1/8 0:0
        昭和最後の日 = 日本標準時.getTime();
    }

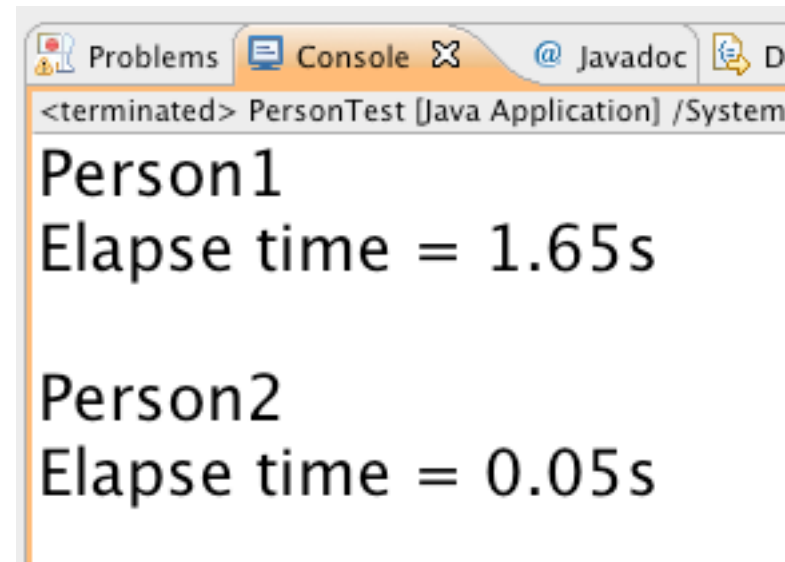
    public boolean 昭和生れですか?() {
        return (birthDate.compareTo(昭和最初の日) >= 0 && birthDate
                .compareTo(昭和最後の日) < 0);
    }
}

```


実験: PersonTest.java

```
6 Random r = new Random();
7
8 private static final int N = 1000000;
9
10 private void test(Person[] persons) {
11     long time = System.currentTimeMillis();
12     for (int i = 0; i < N; i++)
13         persons[r.nextInt(persons.length)].isShowaBaby();
14     System.out.printf("Elapse time = %.2fs\n",
15                       (System.currentTimeMillis() - time) / 1000.0);
16 }
```

実験結果: PersonTest.java



The screenshot shows an IDE console window with tabs for Problems, Console, Javadoc, and Debug Console. The Console tab is active, displaying the output of a Java application. The output shows two objects, Person1 and Person2, with their respective elapsed execution times. Person1 took 1.65 seconds, and Person2 took 0.05 seconds. The window title bar indicates the application is 'PersonTest [Java Application]' running on the 'System' JVM.

```
<terminated> PersonTest [Java Application] /System,
Person1
Elapse time = 1.65s

Person2
Elapse time = 0.05s
```

33倍の効率改善

実験条件

Mac OS X

バージョン 10.7.5

[ソフトウェア・アップデート...](#)

プロセッサ 2.53 GHz Intel Core 2 Duo

メモリ 4 GB 1067 MHz DDR3

まとめ (avoid creation)

- 定数オブジェクトを扱うときは無駄なnewがないことに注意
- static field など簡単に効率が改善できる
- プログラムの可読性が向上することもある
- 場合によっては数十倍の効率改善