

計算機科学第一

2013年度第9回

日付

22: Favor static member classes over nonstatic

4 種の内部クラス

	static member	nonstatic member	annonymous	local
用途	public helper	Adapter	Function Process	クロージャ ヤ？
特徴	○メモリ 効率	enclosing instance ×メモリ効 率	便利だが 制約多数	

本日の流れ

- ❖ Static vs Non-static member classes
- ❖ Non-static member class とその利用
- ❖ Static member class が利用できる場合
- ❖ Anonymous class とその利用
- ❖ Local class とその利用
- ❖ まとめ

Non-static member classの利用

4 種の内部クラス

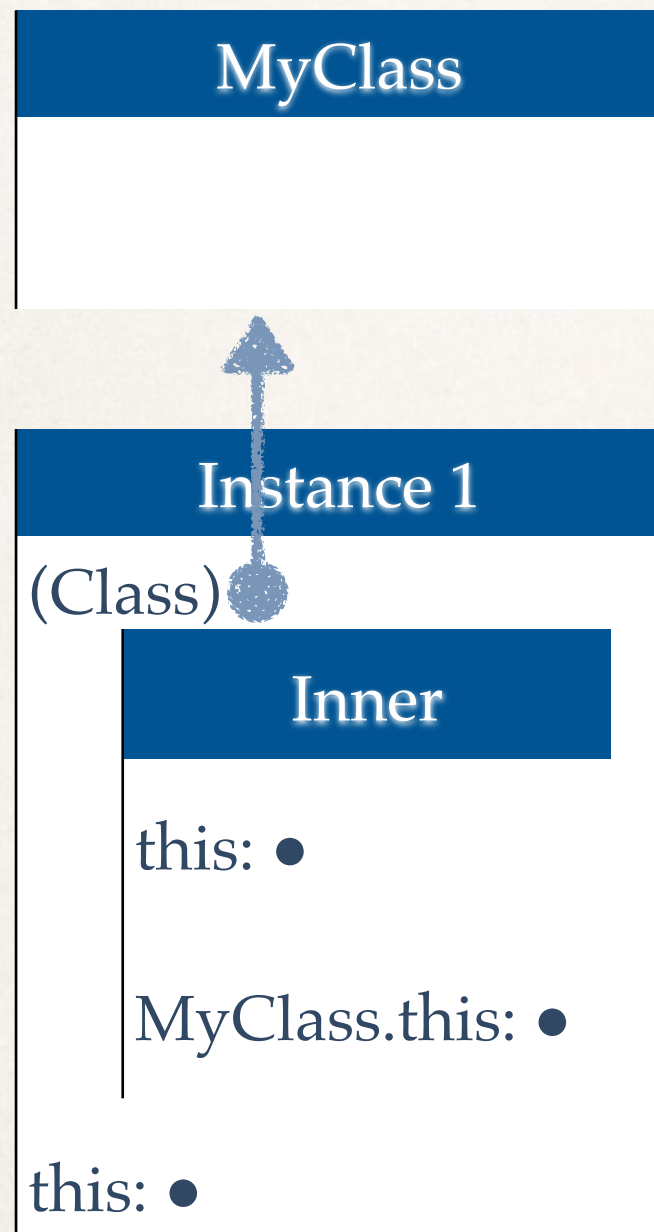
	static member	nonstatic member	annonymous	local
用途	public helper	Adapter	Function Process	クロージャ ヤ？
特徴	○メモリ 効率	enclosing instance ×メモリ効 率	便利だが 制約多数	

Nonstatic member class

```
class Outer {  
    class Inner { // ※ static modifier がない  
        ...  
    }  
    ... new Inner(...)  
}
```

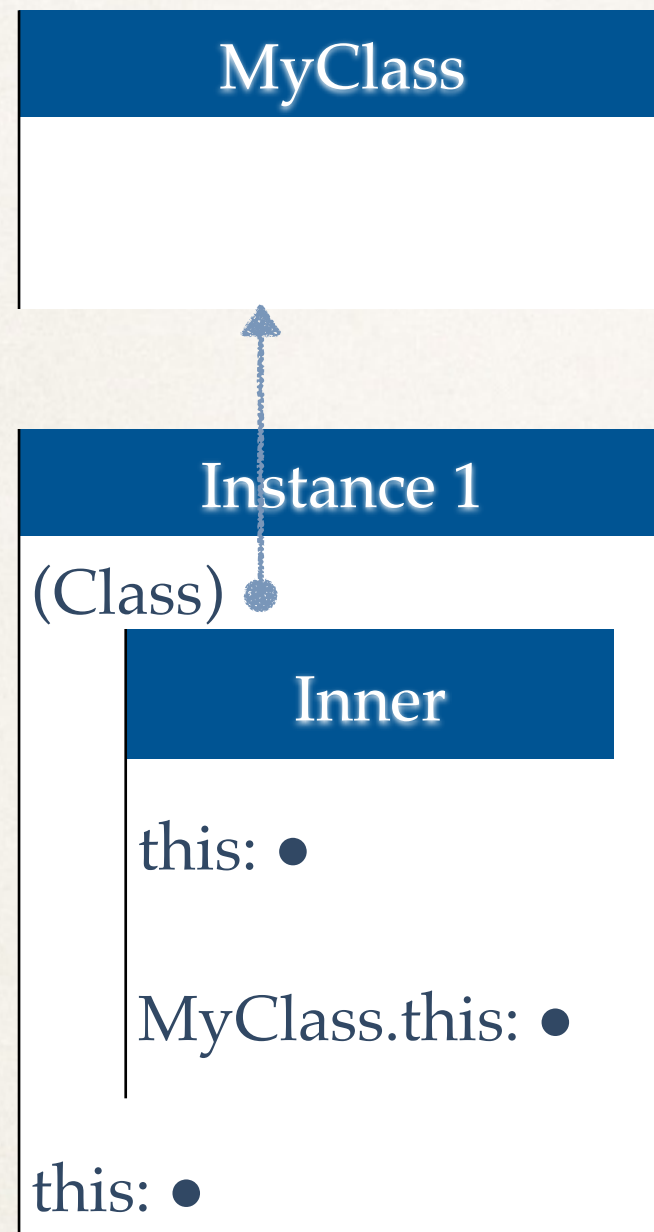
... new Outer.Inner(...) ...

Nonstatic member class



❖ オブジェクトの包含関係

Nonstatic member class



- ❖ オブジェクトの包含関係
- ❖ 外側のオブジェクトのメンバーの参照

NSMCの用例1: Iterator

```
public void fibonacci() {  
    FibonacciList v = new FibonacciList();  
    for (int i = 0; i < 1000; i++) v.add(i);  
  
    for (int i = 0; i < v.size() && i < 10; i++)  
        out.printf("%sv[%d] = %d", i > 0 ? ", " : "", i, v.get(i));  
    out.println();  
  
    System.out.println(v);  
}
```


NSMCの用例1: Iterator

```
public void fibonacci() {  
    FibonacciList v = new FibonacciList();  
    for (int i = 0; i < 1000; i++) v.add(i);  
  
    for (int i = 0; i < v.size() && i < 10; i++)  
        out.printf("%sv[%d] = %d", i > 0 ? ", " : "", i, v.get(i));  
    out.println();  
  
    System.out.println(v);  
}
```

Fibonacci Iterator

- ❖ Fibonacci数-個目の要素のみを走査する（かなり変な）イタレータ

Fibonacciリストの全要素を出力

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

Fibonacciリストの要素をイタレータを用いて出力

v[0] = 1

v[1] = 2

v[2] = 3

v[3] = 5

v[4] = 8

v[5] = 13

v[6] = 21

v[7] = 34

v[8] = 55

v[9] = 89

v[10] = 144

v[11] = 233

v[12] = 377

v[13] = 610

v[14] = 987

NSMCの用例1: Iterator

```
class FibonacciList implements Iterable<Integer> {
    ArrayList<Integer> v = new ArrayList<Integer>();

    public boolean add(int x) { return v.add(x); }
    public void clear() { v.clear(); }
    public int get(int x) { return v.get(x); }
    public int indexOf(int x) { return v.indexOf(x); }
    public boolean isEmpty() { return v.isEmpty(); }
    public int size() { return v.size(); }
    public Iterator<Integer> iterator() { return new Skiplterator(); }

    private class Skiplterator implements Iterator<Integer> {
        int i = 1, j = 1; // Fibonacci 系列の発生
        public boolean hasNext() { return i < v.size(); }
        public Integer next() {
            int result = v.get(i);
            int t = i; i = i + j; j = t;
            return result;
        }
        public void remove() {}
    }
}
```

NSMCの用例1: Iterator

```
class FibonacciList implements Iterable<Integer> {  
    ArrayList<Integer> v = new ArrayList<Integer>();  
  
    public boolean add(int x) { return v.add(x); }  
    public void clear() { v.clear(); }  
    public int get(int x) { return v.get(x); }  
    public int indexOf(int x) { return v.indexOf(x); }  
    public boolean isEmpty() { return v.isEmpty(); }  
    public int size() { return v.size(); }  
    public Iterator<Integer> iterator() { return new Skiplterator(); }  
  
    private class Skiplterator implements Iterator<Integer> {  
        int i = 1, j = 1; // Fibonacci 系列の発生  
        public boolean hasNext() { return i < v.size(); }  
        public Integer next() {  
            int result = v.get(i);  
            int t = i; i = i + j; j = t;  
            return result;  
        }  
        public void remove() {}  
    }  
}
```


NSMCの用例1: Iterator

```
class FibonacciList implements Iterable<Integer> {
    ArrayList<Integer> v = new ArrayList<Integer>();

    public boolean add(int x) { return v.add(x); }
    public void clear() { v.clear(); }
    public int get(int x) { return v.get(x); }
    public int indexOf(int x) { return v.indexOf(x); }
    public boolean isEmpty() { return v.isEmpty(); }
    public int size() { return v.size(); }
    public Iterator<Integer> iterator() { return new Skiplterator(); }

    private class Skiplterator implements Iterator<Integer> {
        int i = 1, j = 1; // Fibonacci 系列の発生
        public boolean hasNext() { return i < v.size(); }
        public Integer next() {
            int result = v.get(i);
            int t = i; i = i + j; j = t;
            return result;
        }
        public void remove() {}
    }
}
```

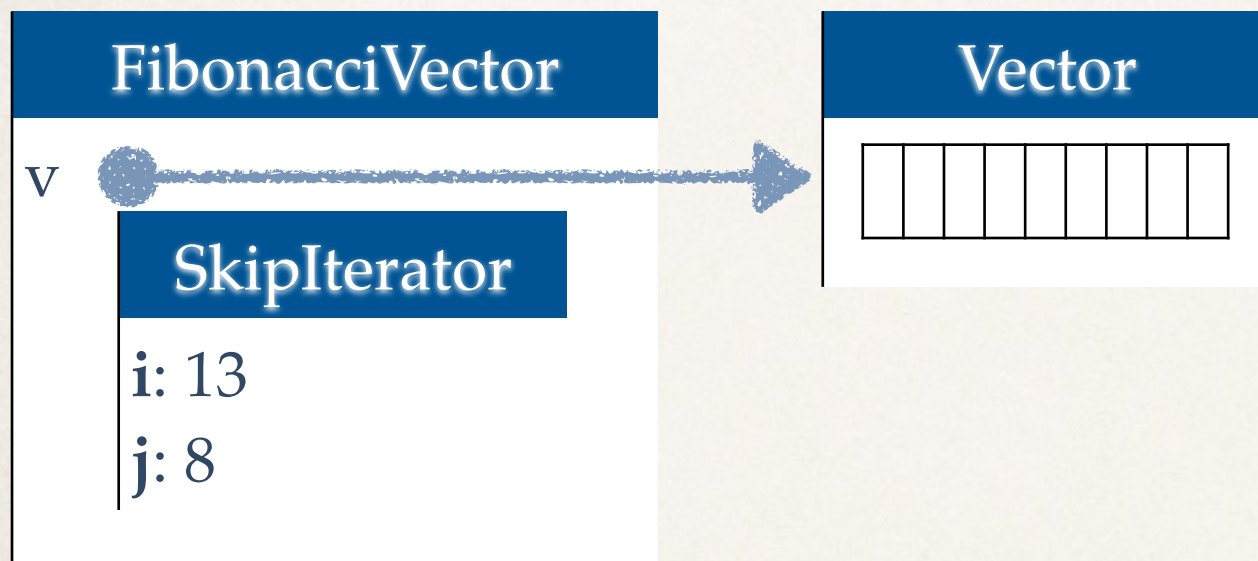
NSMCの用例1: Iterator

```
class FibonacciList implements Iterable<Integer> {  
    ArrayList<Integer> v = new ArrayList<Integer>();  
  
    public boolean add(int x) { return v.add(x); }  
    public void clear() { v.clear(); }  
    public int get(int x) { return v.get(x); }  
    public int indexOf(int x) { return v.indexOf(x); }  
    public boolean isEmpty() { return v.isEmpty(); }  
    public int size() { return v.size(); }  
    public Iterator<Integer> iterator() { return new Skiplterator(); }
```

```
private class Skiplterator implements Iterator<Integer> {  
    int i = 1, j = 1; // Fibonacci 系列の発生  
    public boolean hasNext() { return i < v.size(); }  
    public Integer next() {  
        int result = v.get(i);  
        int t = i; i = i + j; j = t;  
        return result;  
    }  
    public void remove() {}  
}
```


Nonstatic member class

- ❖ オブジェクトの包含関係



- ❖ 外側のオブジェクトのメンバーを参照できる

用例2: Adapter

- ❖ 既存のオブジェクトのクラスを変更せずに、インタフェースを変更する手法
- ❖ $\text{Map} \rightarrow \text{keySet}: \{ k \rightarrow v, \dots \} \rightarrow \{ k \dots \}$
- ❖ $\text{Map} \rightarrow \text{values}: \{ k \rightarrow v, \dots \} \rightarrow \{ v \}$
- ❖ $\text{Map} \rightarrow \text{entrySet}: \{ k \rightarrow v, \dots \} \rightarrow \{ (k, v) \}$

java.util.AbstractMap

メソッドの概要

void	<code>clear()</code> マップからマッピングをすべて削除します (任意のオペレーション)。
protected <code>Object</code>	<code>clone()</code> <code>AbstractMap</code> のインスタンスのシャローコピーを返します。
boolean	<code>containsKey(Object key)</code> マップが指定のキーのマッピングを保持する場合に <code>true</code> を返します。
boolean	<code>containsValue(Object value)</code> マップが1つまたは複数のキーと指定された値をマッピングしている場合に <code>true</code> を返します。
abstract <code>Set<Map.Entry<K, V>></code>	<code>entrySet()</code> このマップに含まれるマップの <code>Set</code> ビューを返します。
boolean	<code>equals(Object o)</code> 指定されたオブジェクトがこのマップと等しいかどうかを比較します。
<code>V</code>	<code>get(Object key)</code> 指定されたキーがマップされている値を返します。
int	<code>hashCode()</code> マップのハッシュコード値を返します。
boolean	<code>isEmpty()</code> マップがキーと値のマッピングを保持しない場合に <code>true</code> を返します。
<code>Set<K></code>	<code>keySet()</code> このマップに含まれるキーの <code>Set</code> ビューを返します。
<code>V</code>	<code>put(K key, V value)</code> 指定された値と指定されたキーをこのマップに関連付けます (任意のオペレーション)。
void	<code>putAll(Map<? extends K, ? extends V> m)</code>

java.util.AbstractMap

❖ Map \rightarrow keySet: $\{ k \rightarrow v, \dots \} \rightarrow \{ k \dots \}$

Set<K>	keySet() このマップに含まれるキーの Set ビューを返します。
------------------------------	---

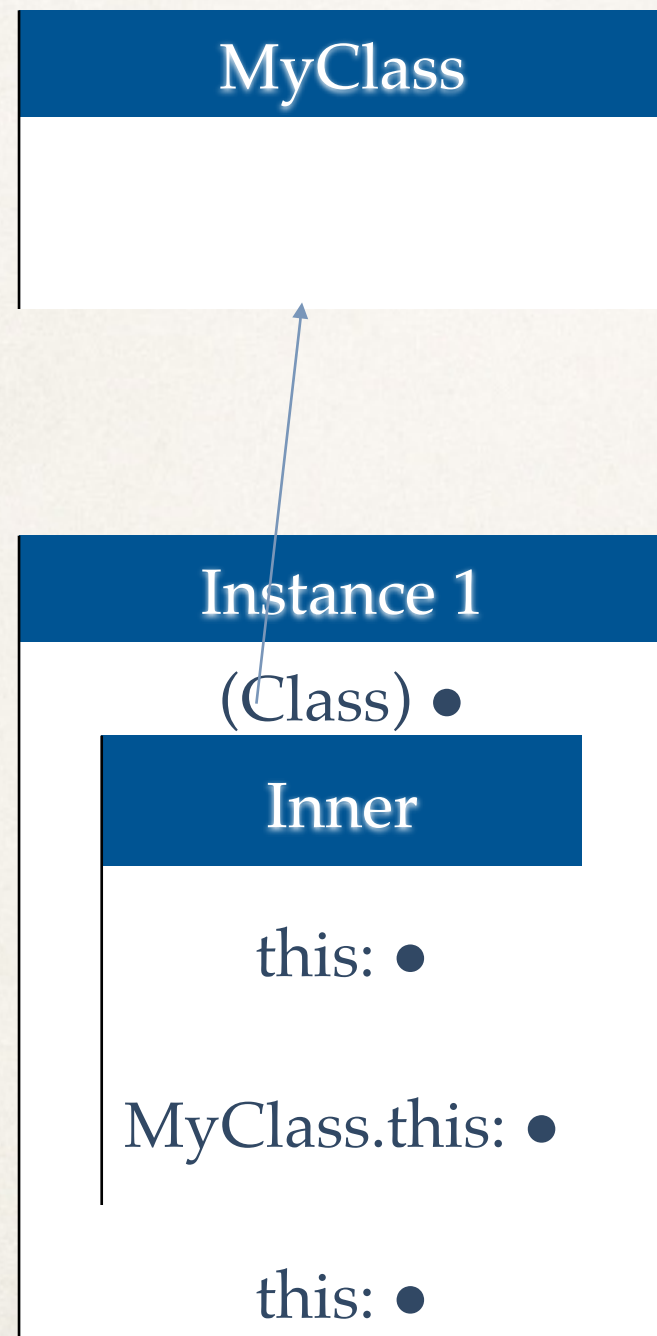
❖ Map \rightarrow values: $\{ k \rightarrow v, \dots \} \rightarrow \{ v \}$

Collection<V>	values() このマップに含まれる値の Collection ビューを返します。
-------------------------------------	---

❖ Map \rightarrow entrySet: $\{ k \rightarrow v, \dots \} \rightarrow \{ (k, v) \}$

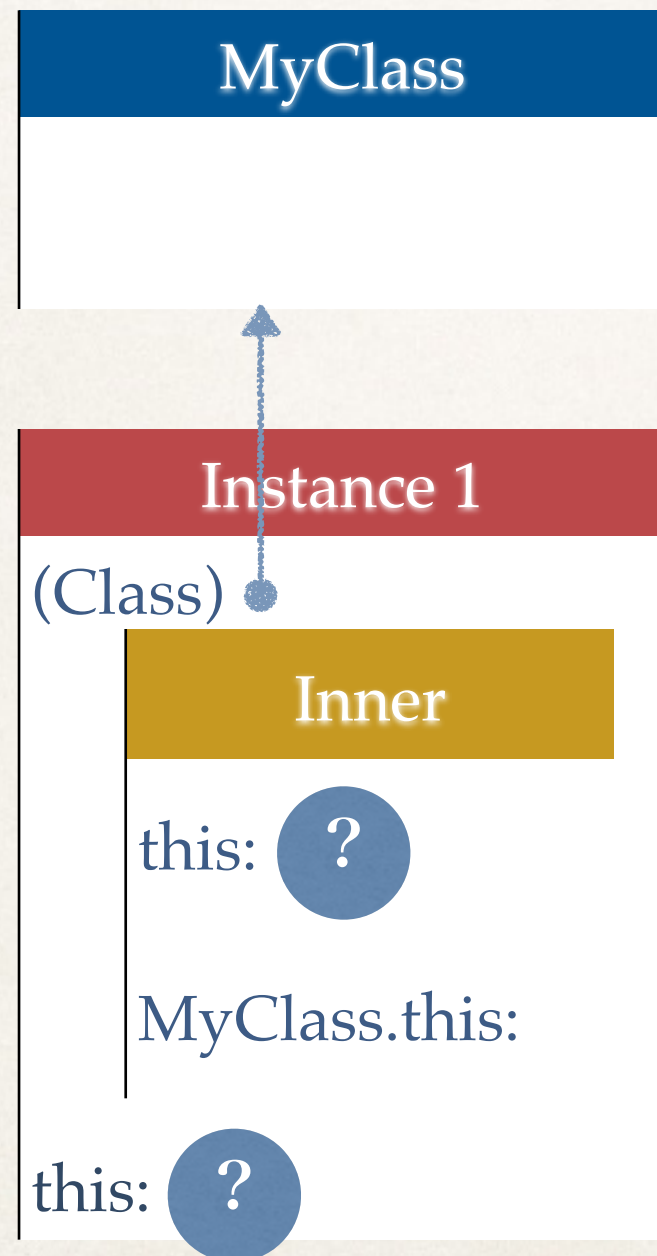
abstract Set<Map.Entry<K,V>>	entrySet() このマップに含まれるマップの Set ビューを返します。
---	--

Nonstatic member class



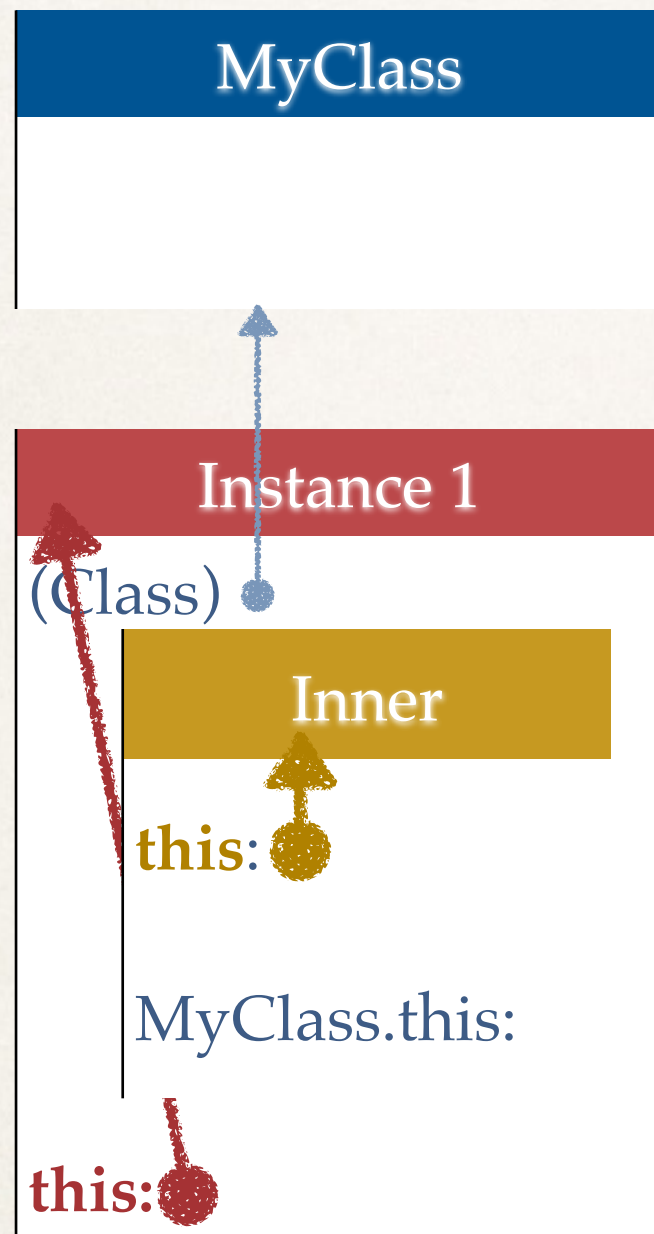
- ❖ オブジェクトの包含関係
- ❖ 外側のオブジェクトのメンバーを参照できる
- ❖ 構造的 `this`

Nonstatic member class



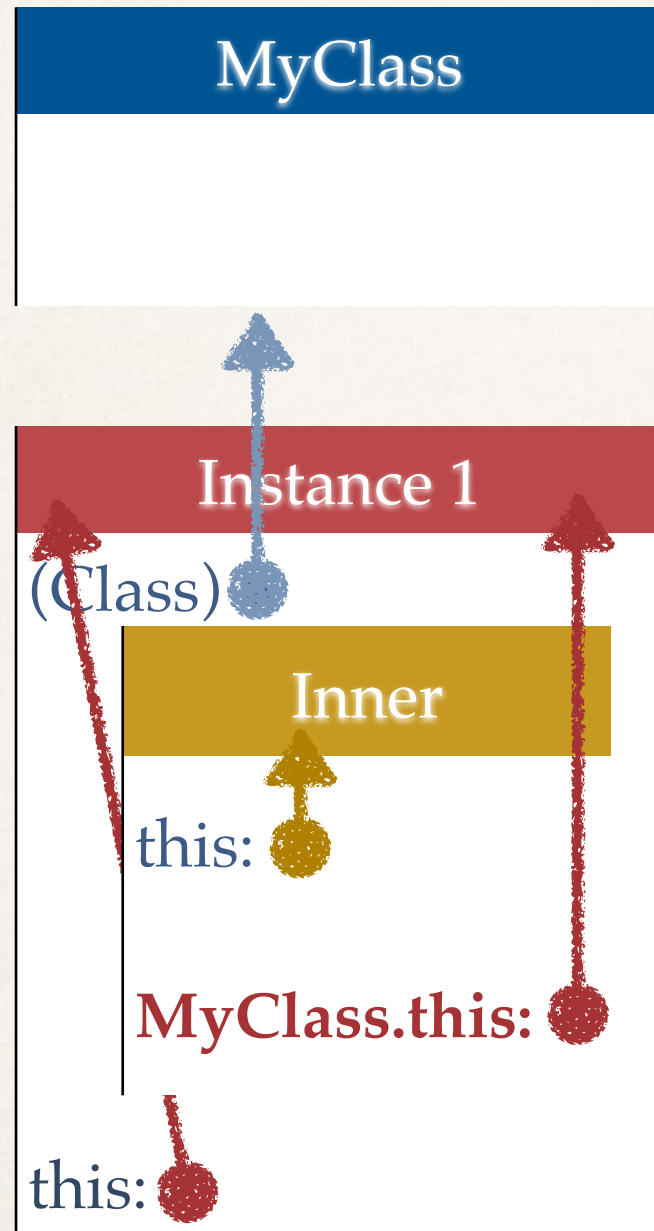
- ❖ オブジェクトの包含関係
- ❖ 外側のオブジェクトのメンバーを参照できる
- ❖ 構造的 this

Nonstatic member class



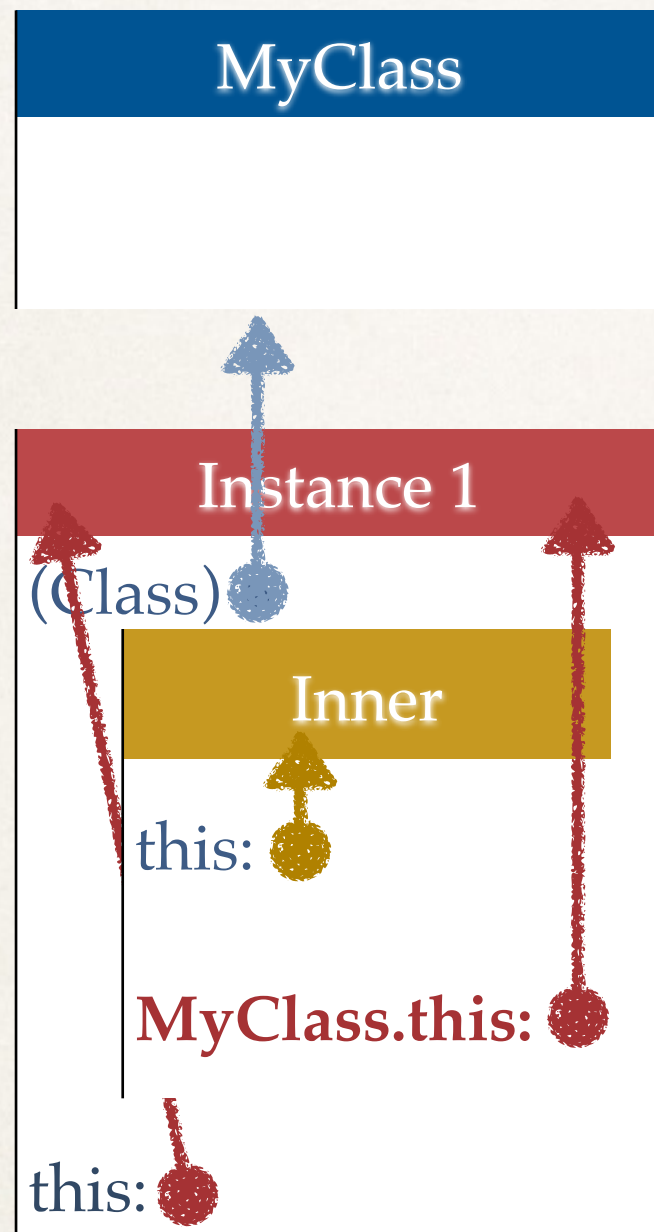
- ❖ オブジェクトの包含関係
- ❖ 外側のオブジェクトのメンバーを参照できる
- ❖ 構造的 this

Nonstatic member class



- ❖ オブジェクトの包含関係
- ❖ 外側のオブジェクトのメンバーを参照できる
- ❖ 構造的 this

Nonstatic member class



- ❖ オブジェクトの包含関係

- ❖ 外側のオブジェクトのメンバーを参照できる

- ❖ 構造的 this

注：super に似て異なるもの

super: 継承関係の親クラスを参照

構造的this: 包含関係の外側のインスタンスを参照

Static member classの利用

4 種の内部クラス

	static member	nonstatic member	annonymous	local
用途	public helper	Adapter	Function Process	クロージャ ヤ？
特徴	○メモリ 効率	enclosing instance ×メモリ効 率	便利だが 制約多数	

Static member class

```
class Outer {  
    static class Inner { // static 宣言されている  
        ...  
    }  
    ... new Inner(...)  
}  
  
... new Outer.Inner(...) ...
```


Static member class

```
class Outer {  
    static class Inner { // static 宣言されている  
        ...  
    }  
    ... new Inner(...)  
}
```

... new Outer.Inner(...) ...

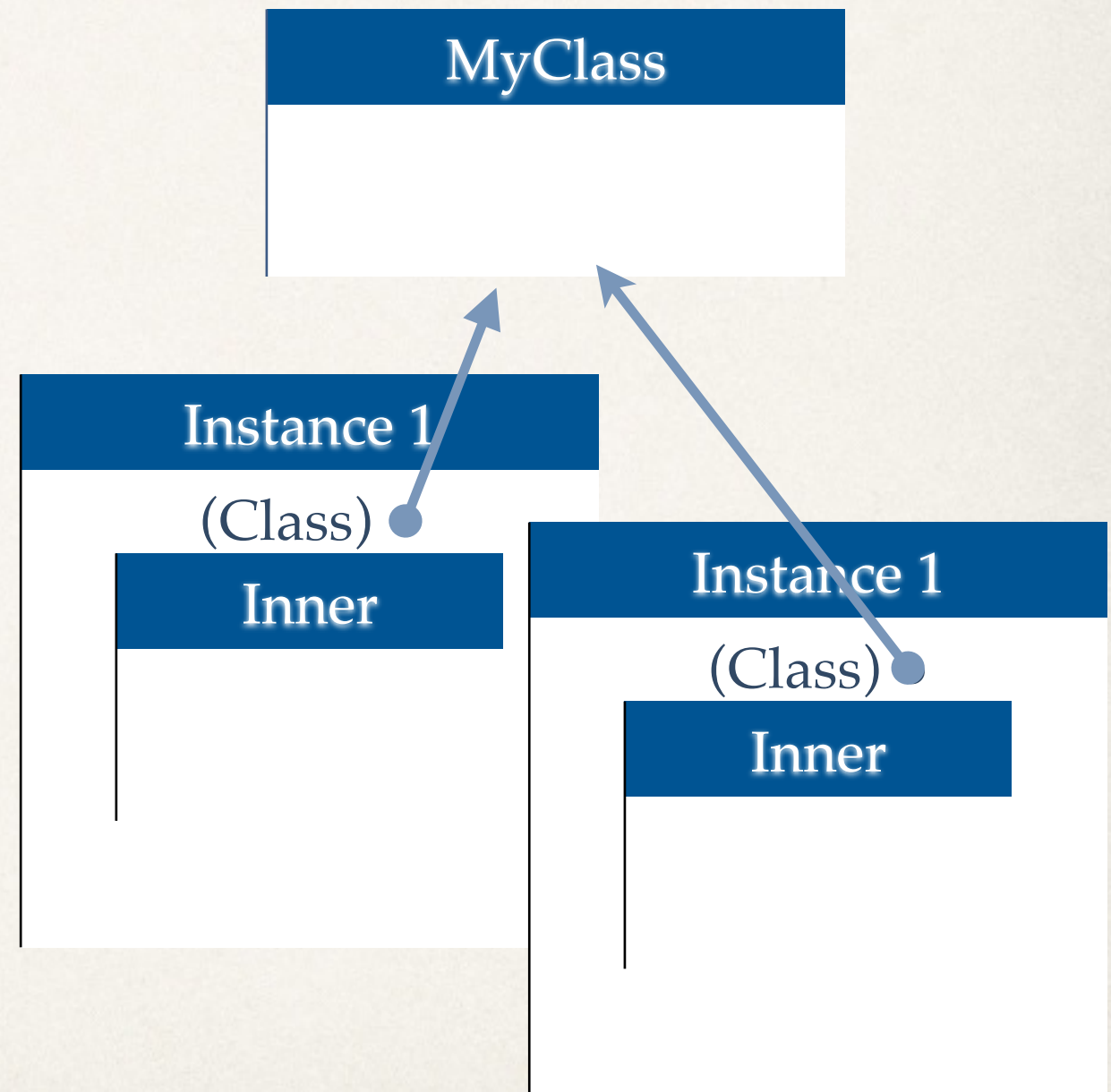
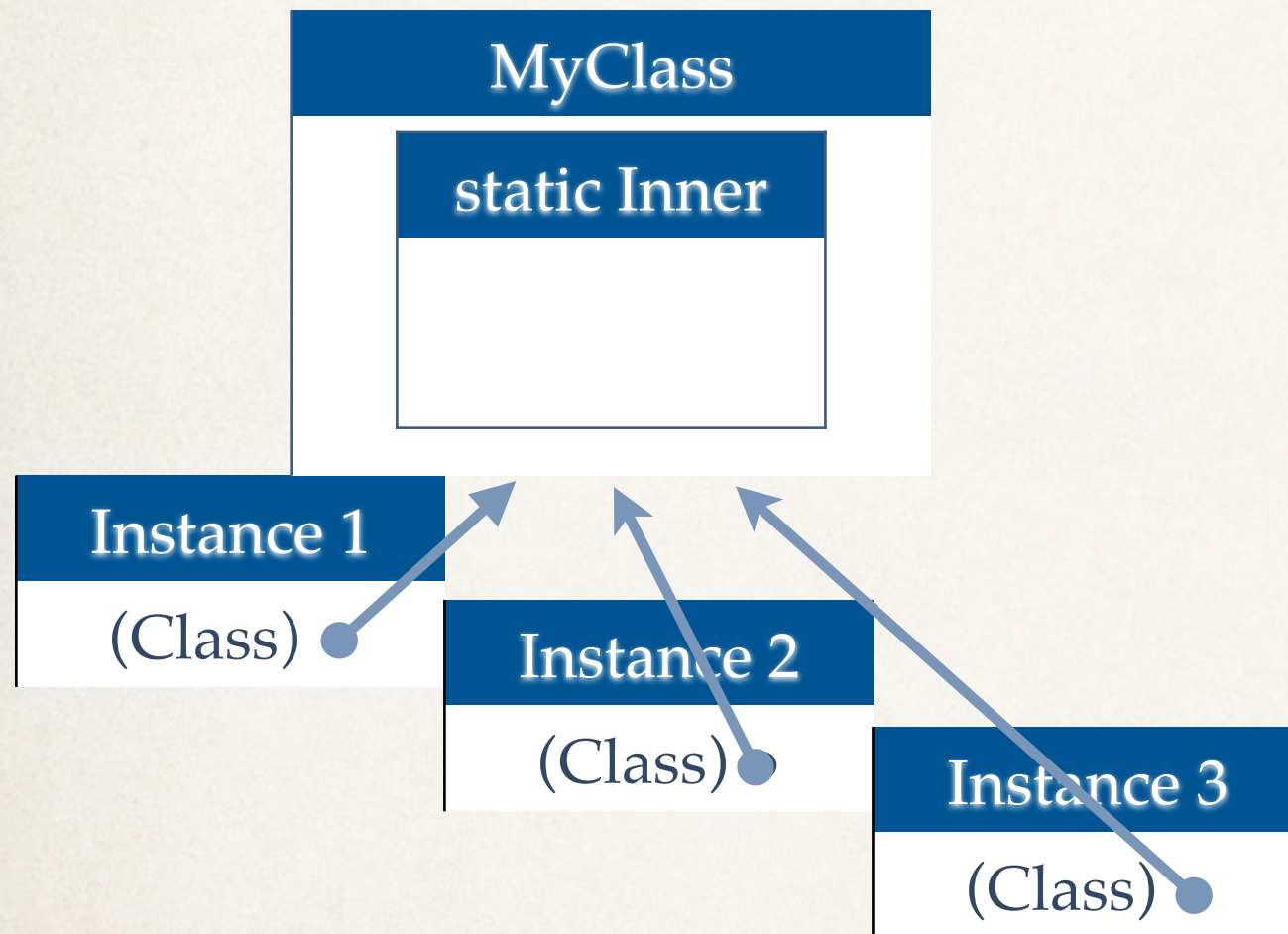
Nonstatic member class

```
class Outer {  
    class Inner { // ※ static modifier がない  
        ...  
    }  
    ... new Inner(...)  
}  
  
... new Outer.Inner(...) ...
```


Static Member Class

vs

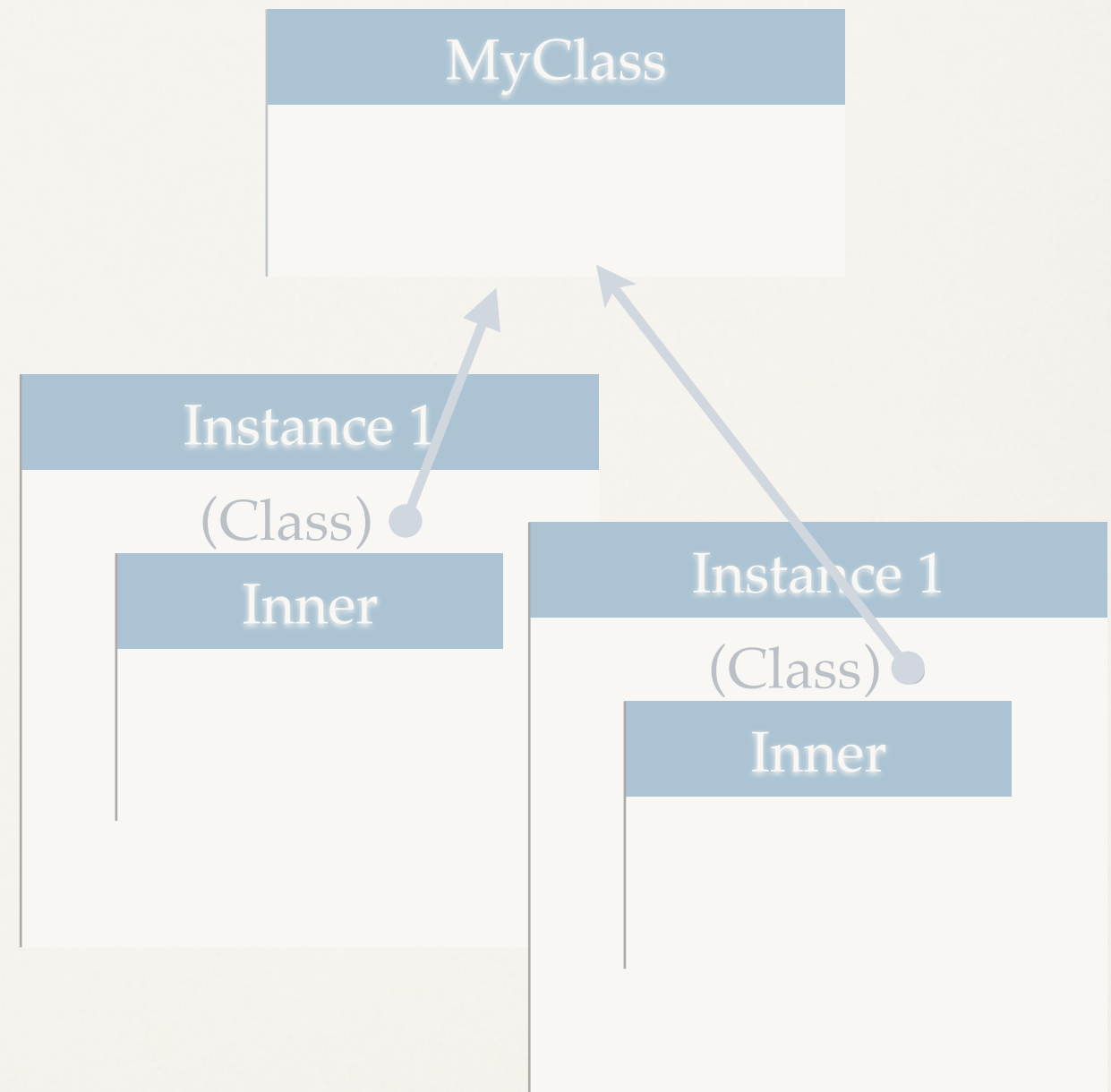
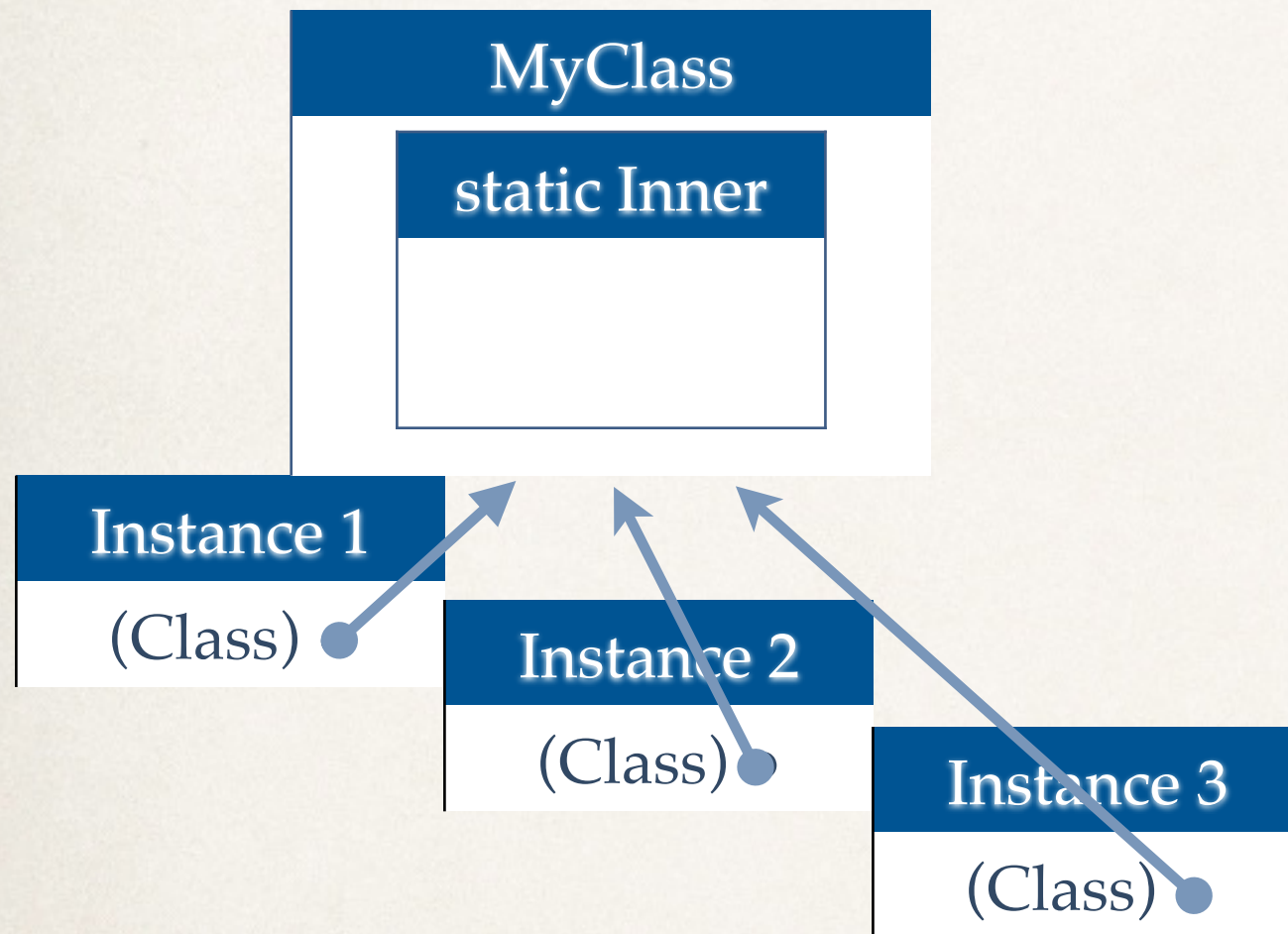
Nonstatic Member Class



Static Member Class

vs

Nonstatic Member Class



外側のobjectをアクセスしない？

- ❖ `static class { ... }`
`static member class` でメモリを節約
- ❖ ※ Java 1.6 では、その差が見えない気がする。。。
`SignalsA vs SignalsB; SignalsC vs SignalsD` でほとんど差がでません

Anonymous class の利用

4 種の内部クラス


	static member	nonstatic member	annonymous (無名)	local
用途	public helper	Adapter	Function Process	クロージャ ヤ？
特徴	○メモリ 効率	enclosing instance ×メモリ効 率	便利だが 制約多数	

Anonymous (無名) class

```
new Object () {  
    フィールドの定義;  
  
    ...  
    メソッドの定義;  
  
    ...  
}
```


Anonymous class

Object クラスを拡張した何か



```
new Object () {  
    フィールドの定義;  
  
    ...  
    メソッドの定義;  
  
    ...  
}
```

Anonymous Class の例

```
private void run() {  
    Object[] objects = new Object[] {  
        new Object(),  
  
        new Object() {  
            public String toString() { return "我輩はオブジェクトである。名前はまだない。"; }  
        }  
    };  
    out.println(Arrays.toString(objects));  
}
```


Anonymous Class の例

```
private void run() {  
    Object[] objects = new Object[] {  
        new Object(),  
  
        new Object() {  
            public String toString() { return "我輩はオブジェクトである。名前はまだない。"; }  
        }  
    };  
    out.println(Arrays.toString(objects));  
}
```

Anonymous Class の例

Problems Tasks Console ScalaTest JUnit

<terminated> AnonymousExample1 [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (2013/12/03 11:19:08)

[java.lang.Object@272d7a10, 我輩はオブジェクトである。名前はまだない。]

Anonymous Class の例

Problems Tasks Console ScalaTest JUnit

<terminated> AnonymousExample1 [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (2013/12/03 11:19:08)

[java.lang.Object@272d7a10, 我輩はオブジェクトである。名前はまだない。]

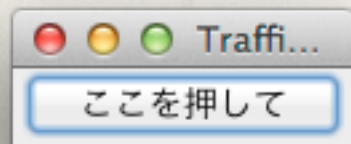
Anonymous class の用例

- ❖ Function object
- ❖ Process object

Function Object

```
private AnonymousClassAsFunctionExample() {  
    super("Traffic Signal");  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    Container pane = getContentPane();  
  
    JButton b = new JButton("ここを押して");  
    b.addMouseListener(new MouseInputAdapter() {  
        public void mouseClicked(MouseEvent e) {  
            System.out.println("こんにちは");  
        }  
    });  
    pane.add(b);  
    this.pack();  
    this.setVisible(true);  
}
```

Function Object



```
private AnonymousClassAsFunctionExample() {  
    super("Traffic Signal");  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    Container pane = getContentPane();  
  
    JButton b = new JButton("ここを押して");  
    b.addMouseListener(new MouseInputAdapter() {  
        public void mouseClicked(MouseEvent e) {  
            System.out.println("こんにちは");  
        }  
    });  
    pane.add(b);  
    this.pack();  
    this.setVisible(true);  
}
```


Process Object

```
private void run() {  
    new Thread() { // スレッド：並列実行を表現するオブジェクト  
        public void run() {  
            while (true) {  
                out.println("生地ができたよ！");  
                try { sleep(1900); } catch (InterruptedException e) {}  
            }  
        }  
    }.start();  
  
    new Thread() { // もうひとつスレッドを用意して2並列  
        public void run() {  
            while (true) {  
                out.println("パンが焼けたよ！");  
                try { sleep(2900); } catch (InterruptedException e) {}  
            }  
        }  
    }.start();  
}
```

Anonymous classの制約

- ❖ new できない
- ❖ instanceof できない
- ❖ implements は高々一つのみ
- ❖ extends + implements もだめ
- ❖ supertype のメンバーしか参照できない
- ❖ (制約ではないが) 短く書くこと

Anonymous classの制約

- ❖ new できない
- ❖ instanceof できない
- ❖ implements は高々一つのみ
- ❖ extends + implements もだめ
- ❖ supertype のメンバーしか参照できない
- ❖ (制約ではないが) 短く書くこと

Anonymous classの制約

- ❖ new できない
- ❖ instanceof できない
- ❖ implements は高々一つのみ
- ❖ extends + implements もだめ
- ❖ supertype のメンバーしか参照できない
- ❖ (制約ではないが) 短く書くこと

Anonymous classの制約

- ❖ new できない
- ❖ instanceof できない
- ❖ implements は高々一つのみ
- ❖ extends + implements もだめ
- ❖ supertype のメンバーしか参照できない
- ❖ (制約ではないが) 短く書くこと

Anonymous classの特徴

- ❖ 実行時にクラスを生成
- ❖ 式が書けるところであればどこでも
- ❖ nonstaticな文脈のときに enclosing
- ❖ static member は持てない（名前がないから）

Local class

- ❖ Anonymous class に名前を与えたもの
- ❖ Anonymous class の制約を嫌う場合に適宜利用
 - ❖ isinstance とか

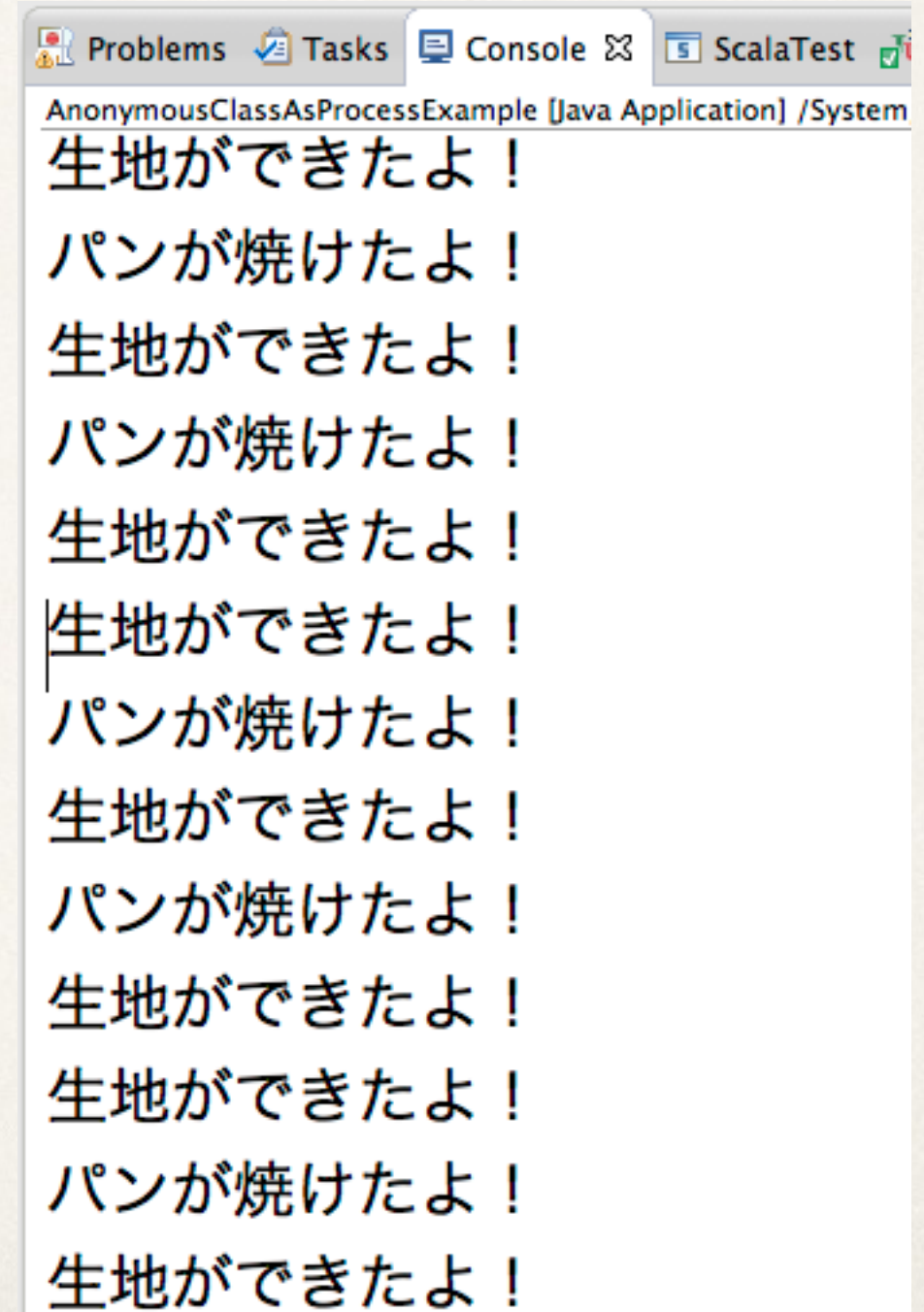
その他のプログラム例 (16個)

Signals1, Signals2, ..., SignalsD

1: 素朴	5: 内部クラス	9: static member クラス
2: 素朴	6: 抽象内部クラス	A: 大量のボタン
3: 抽象クラス	7: 冗長性除去 (ループ)	B: 大量のボタン
4: MIAdapter	8: 局所クラス	C-D: ベンチマーク

実行例

- ❖ パン生地: 1.9秒ごと
- ❖ パン焼き: 2.9秒ごと
- ❖ 配達: 3.7秒ごと



```
AnonymousClassAsProcessExample [Java Application] /System
生地ができたよ！
パンが焼けたよ！
生地ができたよ！
パンが焼けたよ！
生地ができたよ！
生地ができたよ！
パンが焼けたよ！
生地ができたよ！
パンが焼けたよ！
生地ができたよ！
生地ができたよ！
パンが焼けたよ！
生地ができたよ！
```