

# 計算機科學第一

2012年度第8回

---

2012.12.11

# もくじ

---

♣22: Favor static member classes over nonstatic

# 22: Favor static member classes over nonstatic

---

# 4種の入れ子クラス

---

# 4種の内部クラス

	static member	nonstatic member	anonymous	local
用途	public helper	Adapter	Function Process	クロージャ？
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 {  
        ...  
    }  
    ... new Inner(...)  
}  
  
... new Outer.Inner(...) ...
```

# Static member class

---

```
class Outer {  
    static class Inner {  
        ...  
    }  
    ... new Inner(...)  
}
```

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

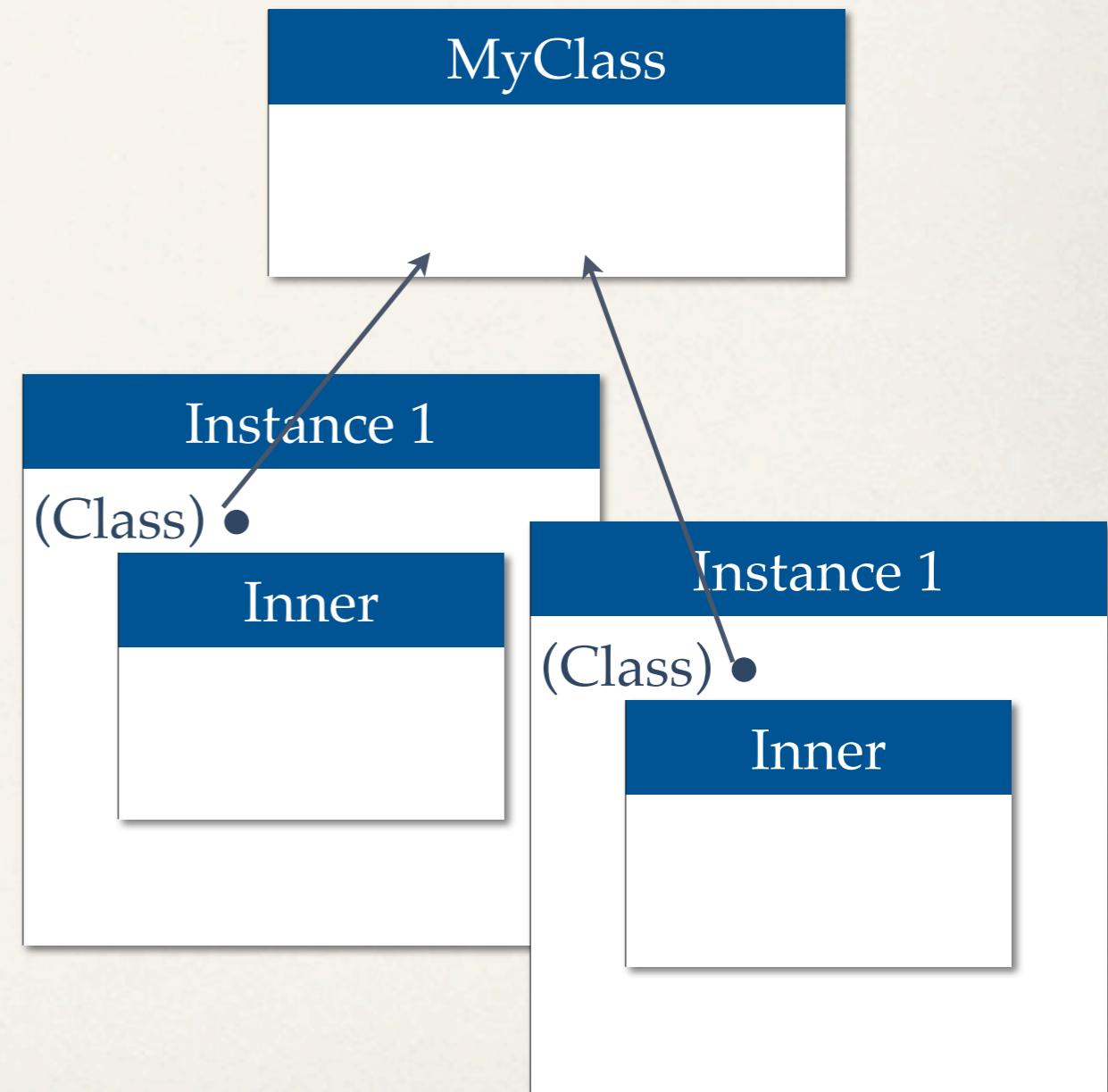
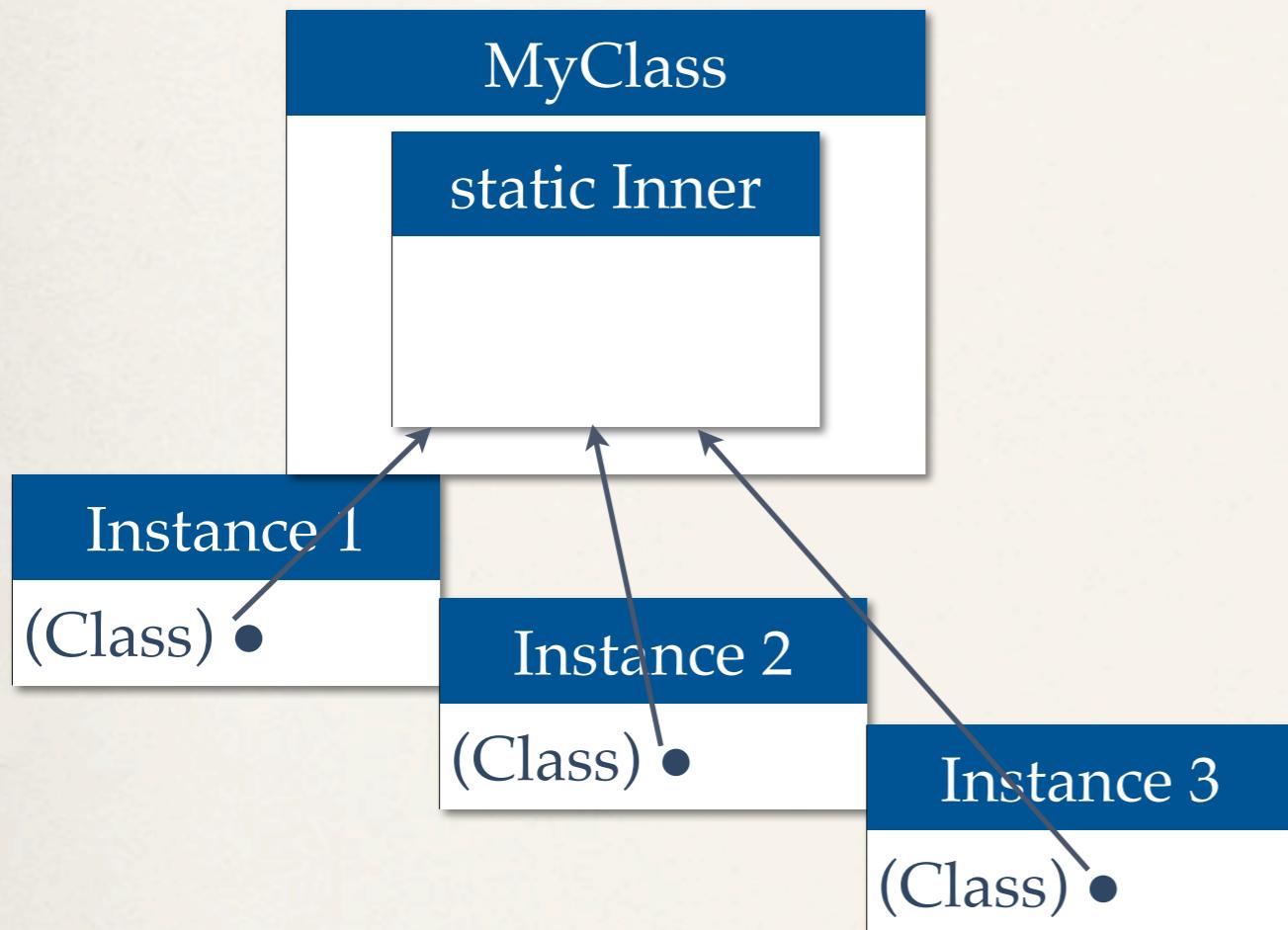
# Nonstatic member class

---

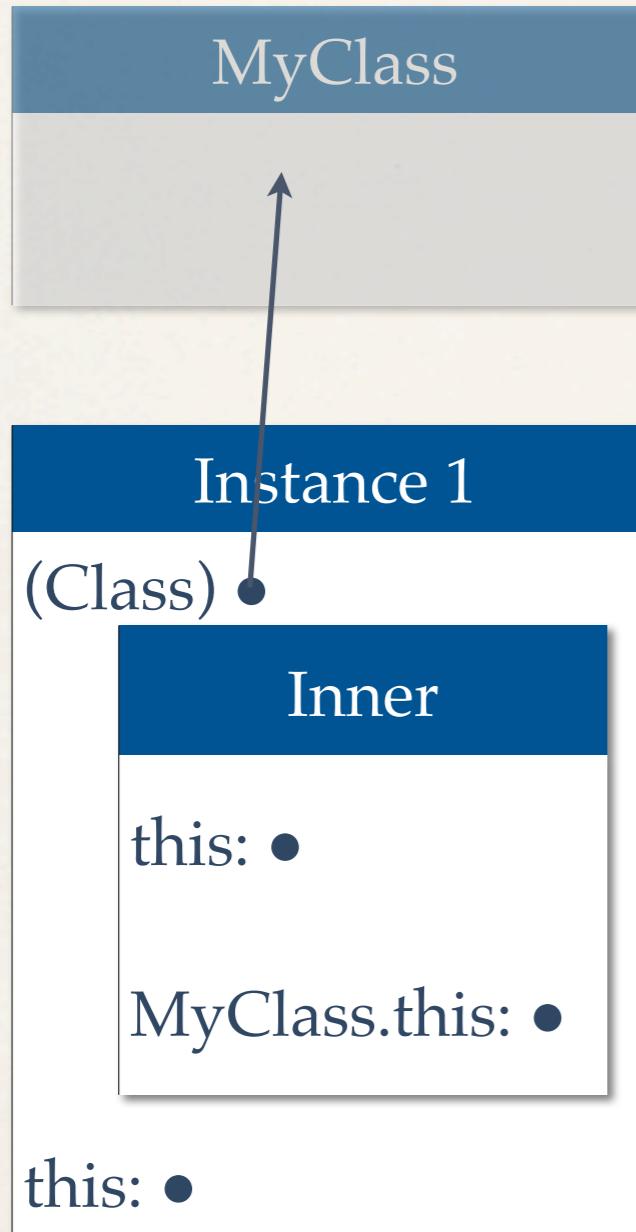
```
class Outer {  
    class Inner { // ※ static modifier がない  
        ...  
    }  
    ... new Inner(...)  
}
```

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

# Static vs Nonstatic Member Class

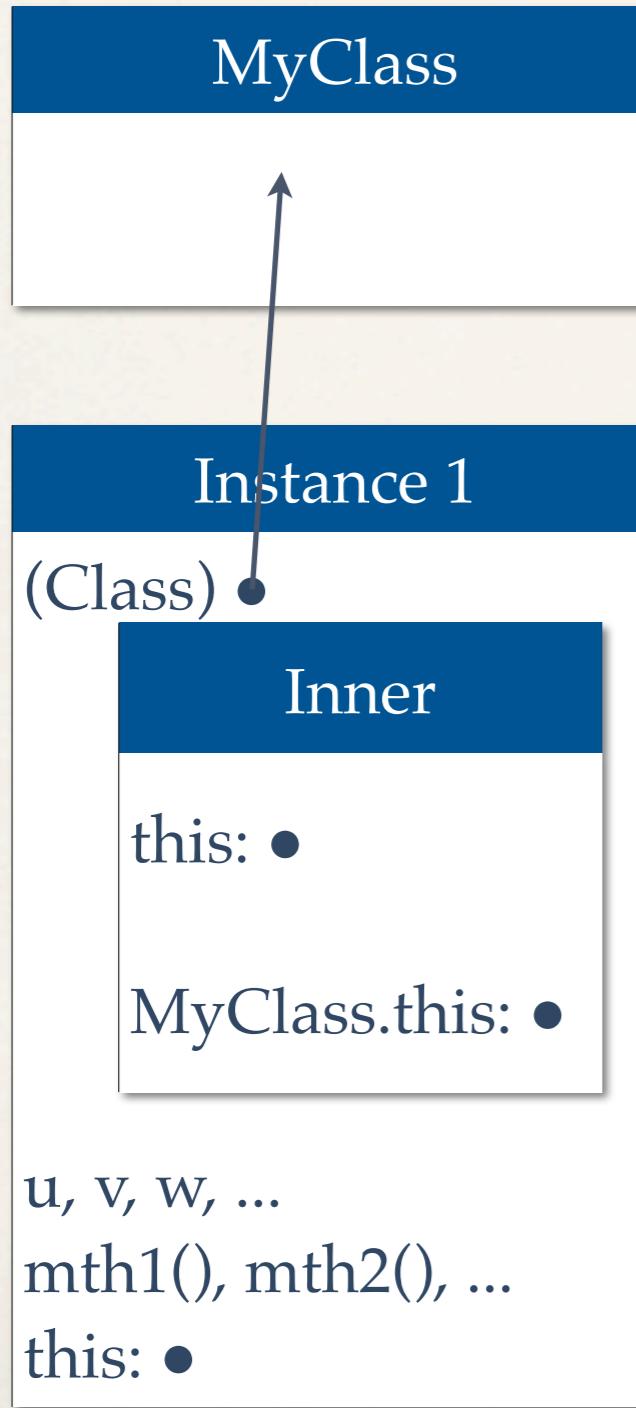


# Nonstatic member class



- 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

# 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();

    boolean head = true;
    for (int n : v) {
        System.out.printf("%s%d", head ? "" : ", ", n);
        head = false;
    }
    out.println();
}
```

# 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();

    boolean head = true;
    for (int n : v) {
        System.out.printf("%s%d", head ? "" : ", ", n);
        head = false;
    }
    out.println();
}
```

# Fibonacci Iterator

---

- Fibonacci数-個目の要素のみを走査する（かなり変な）  
iterator

1	v[0] = 0
2	v[25] = 25
3	v[50] = 50
5	v[75] = 75
8	v[100] = 100
13	...
21	
34	
55	
89	
144	
233	
377	
610	
987	

# 用例1: Iterator

```
class FibonacciVector implements Iterable<Integer> {
    List<Integer> v = new Vector<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 SkipIterator(); }

    private class SkipIterator implements Iterator<Integer> {
        int i = 1, j = 1;
        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() {}
    }
}
```

# 用例1: Iterator

```
class FibonacciVector implements Iterable<Integer> {
    List<Integer> v = new Vector<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 SkipIterator(); }

    private class SkipIterator implements Iterator<Integer> {
        int i = 1, j = 1;
        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() {}
    }
}
```

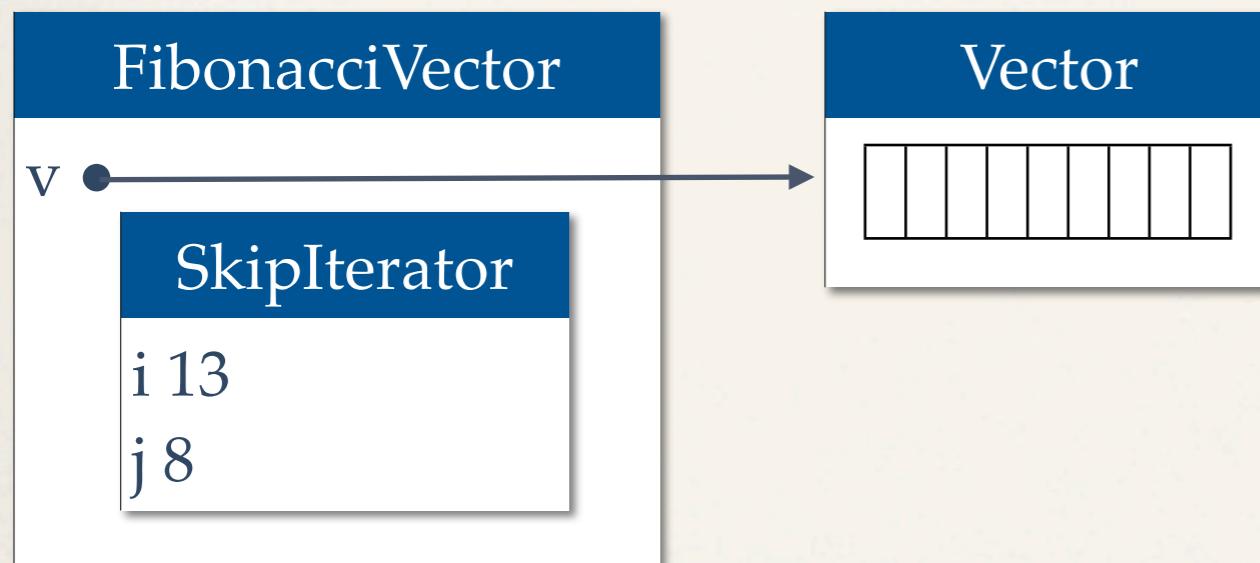
# 用例1: Iterator

```
class FibonacciVector implements Iterable<Integer> {
    List<Integer> v = new Vector<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 SkipIterator(); }

    private class SkipIterator implements Iterator<Integer> {
        int i = 1, j = 1;
        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の参照

# 用例2: Adapter

---

- 既存のオブジェクトのクラスを変更せずに、インターフェイスを変更する手法
  - Map → keySet: { k → v, ... } → { k ... }
  - Map → values: { k → v, ... } → { v }
  - Map → entrySet: { k → v, ... } → { (k, v) }

# java.util.AbstractMap

## メソッドの概要

void	<a href="#">clear()</a> 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 Object	<a href="#">clone()</a> AbstractMap のインスタンスのシャローコピーを返します。
boolean	<a href="#">containsKey(Object key)</a> マップが指定のキーのマッピングを保持する場合に true を返します。
boolean	<a href="#">containsValue(Object value)</a> 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 true を返します。
abstract <a href="#">Set&lt;Map.Entry&lt;K,V&gt;&gt;</a>	<a href="#">entrySet()</a> このマップに含まれるマップの <a href="#">Set</a> ビューを返します。
boolean	<a href="#">equals(Object o)</a> 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。
V	<a href="#">get(Object key)</a> 指定されたキーがマップされている値を返します。
int	<a href="#">hashCode()</a> マップのハッシュコード値を返します。
boolean	<a href="#">isEmpty()</a> マップがキーと値のマッピングを保持しない場合に true を返します。
<a href="#">Set&lt;K&gt;</a>	<a href="#">keySet()</a> このマップに含まれるキーの <a href="#">Set</a> ビューを返します。
V	<a href="#">put(K key, V value)</a> 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	<a href="#">putAll(Map&lt;? extends K,? extends V&gt; m)</a>

# java.util.AbstractMap

---

- \* Map → keySet: { k → v, ... } → { k ... }

<code>Set&lt;K&gt;</code>	<code>keySet()</code>
---------------------------	-----------------------

このマップに含まれるキーの `Set` ビューを返します。

- \* Map → values: { k → v, ... } → { v }

<code>Collection&lt;V&gt;</code>	<code>values()</code>
----------------------------------	-----------------------

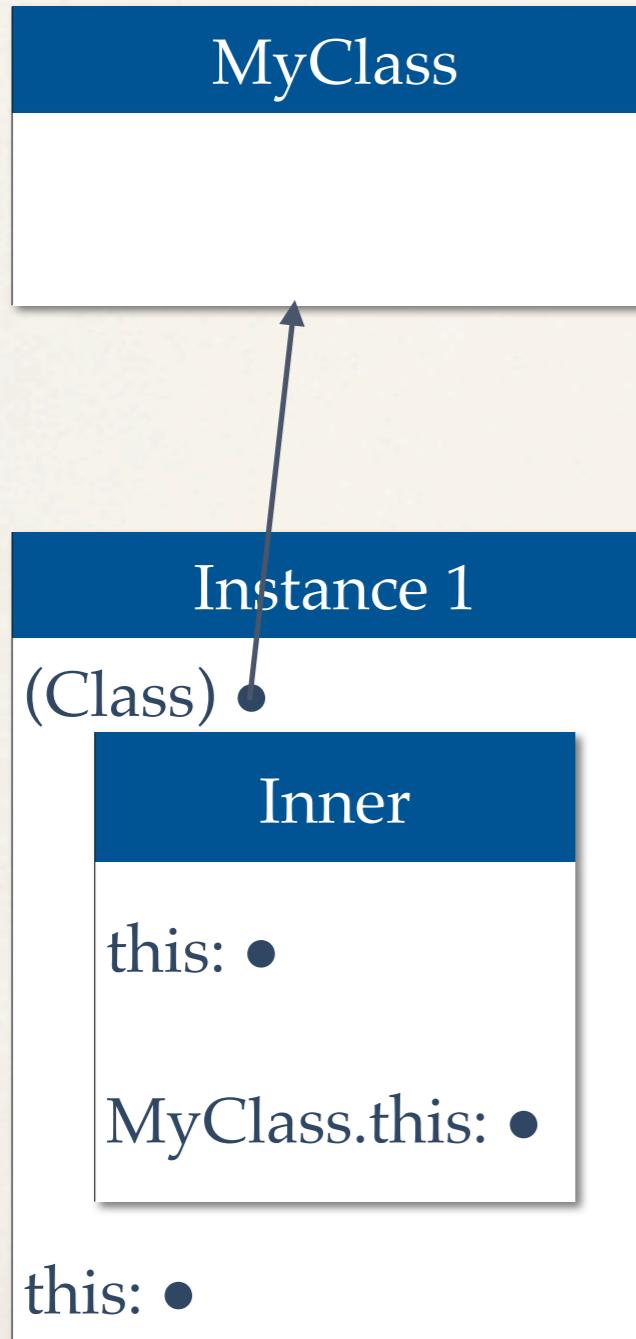
このマップに含まれる値の `Collection` ビューを返します。

- \* Map → entrySet: { k → v, ... } → { (k, v) }

abstract <code>Set&lt;Map.Entry&lt;K,V&gt;&gt;</code>	<code>entrySet()</code>
--	-------------------------

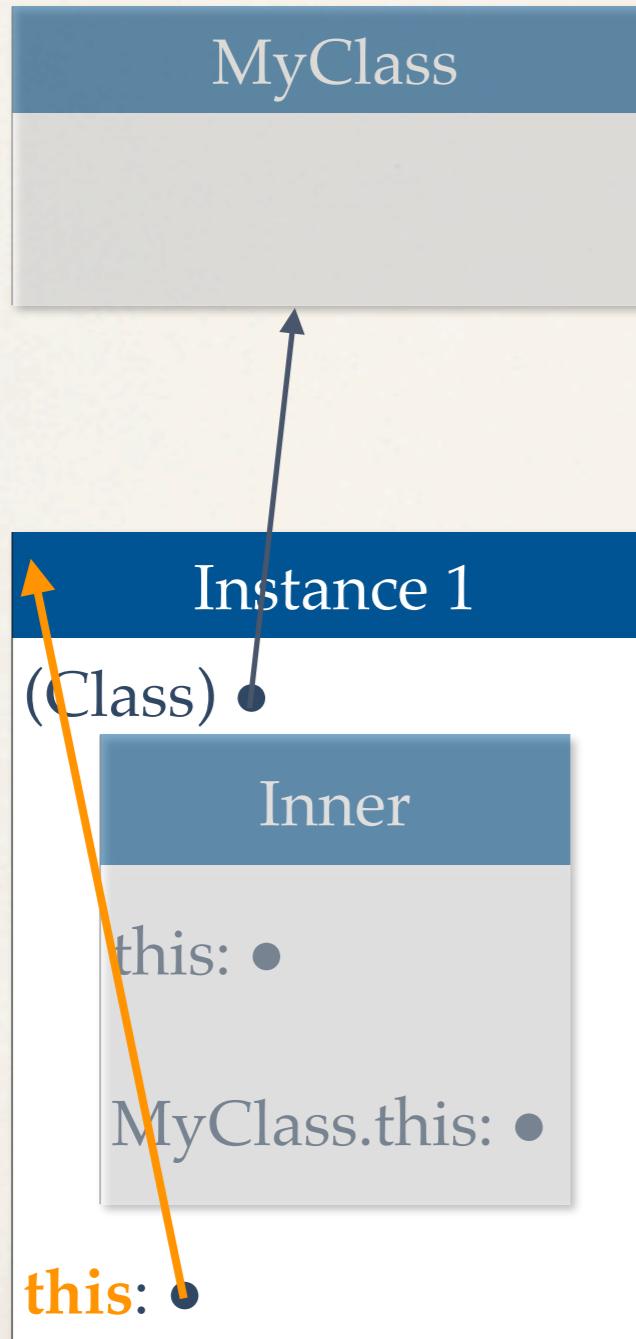
このマップに含まれるマップの `Set` ビューを返します。

# Nonstatic member class



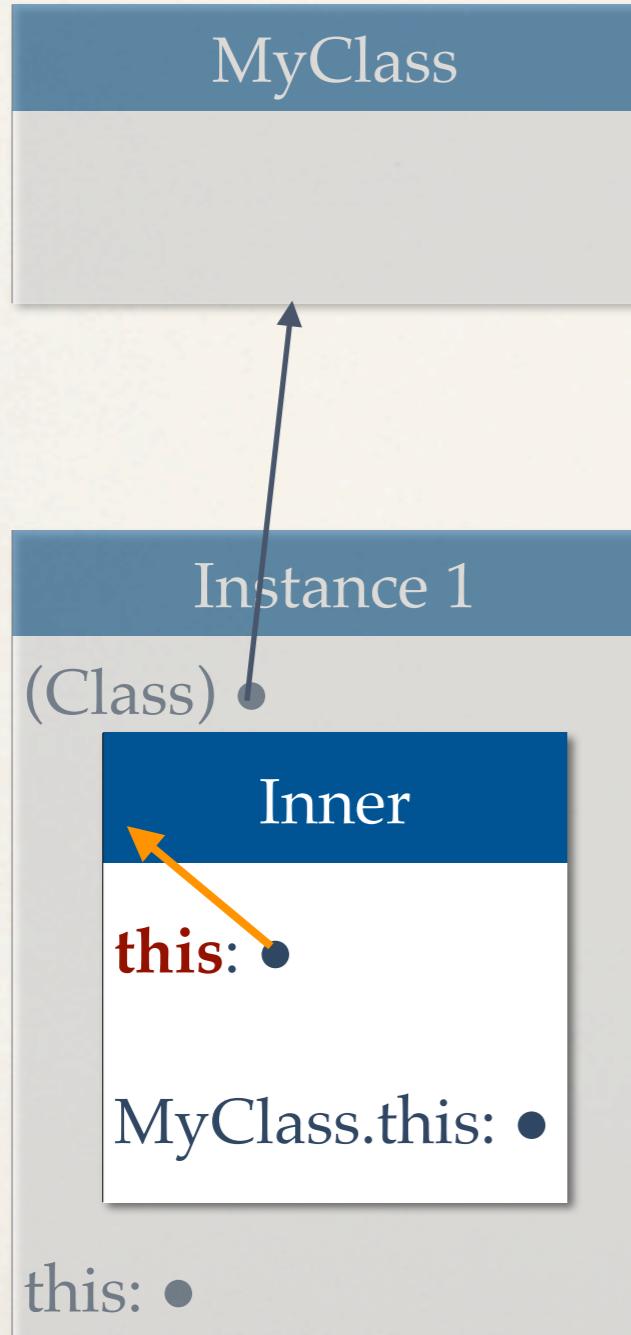
- オブジェクトの包含関係
- 外側のオブジェクトのメンバーの参照
- 構造的 this

# Nonstatic member class



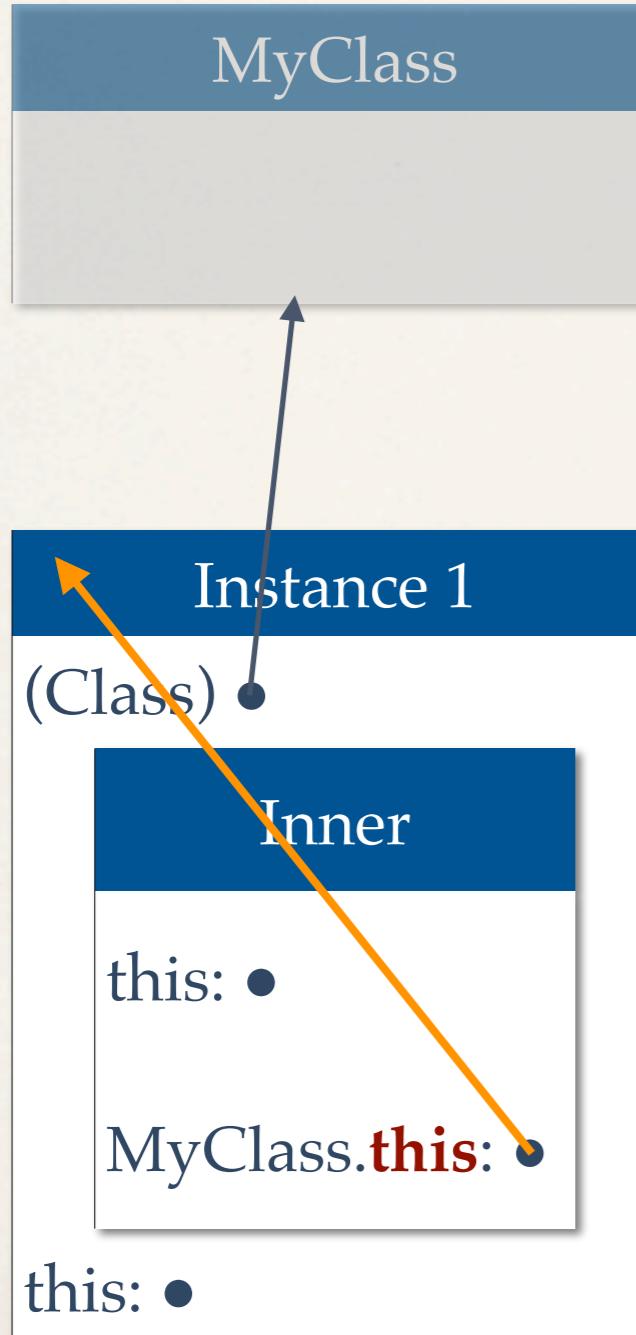
- オブジェクトの包含関係
- 外側のオブジェクトのメンバーの参照
- 構造的 `this`

# Nonstatic member class



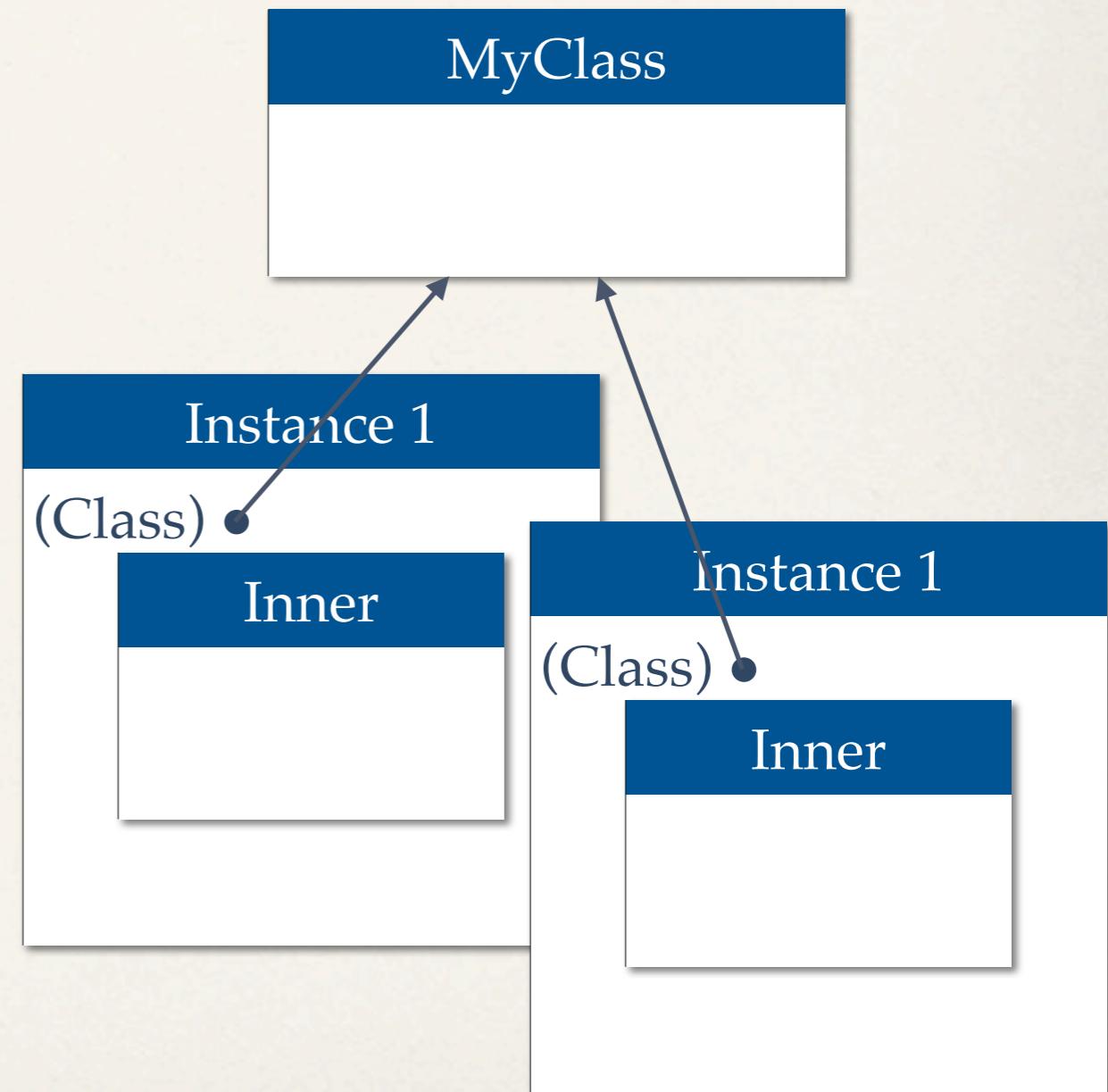
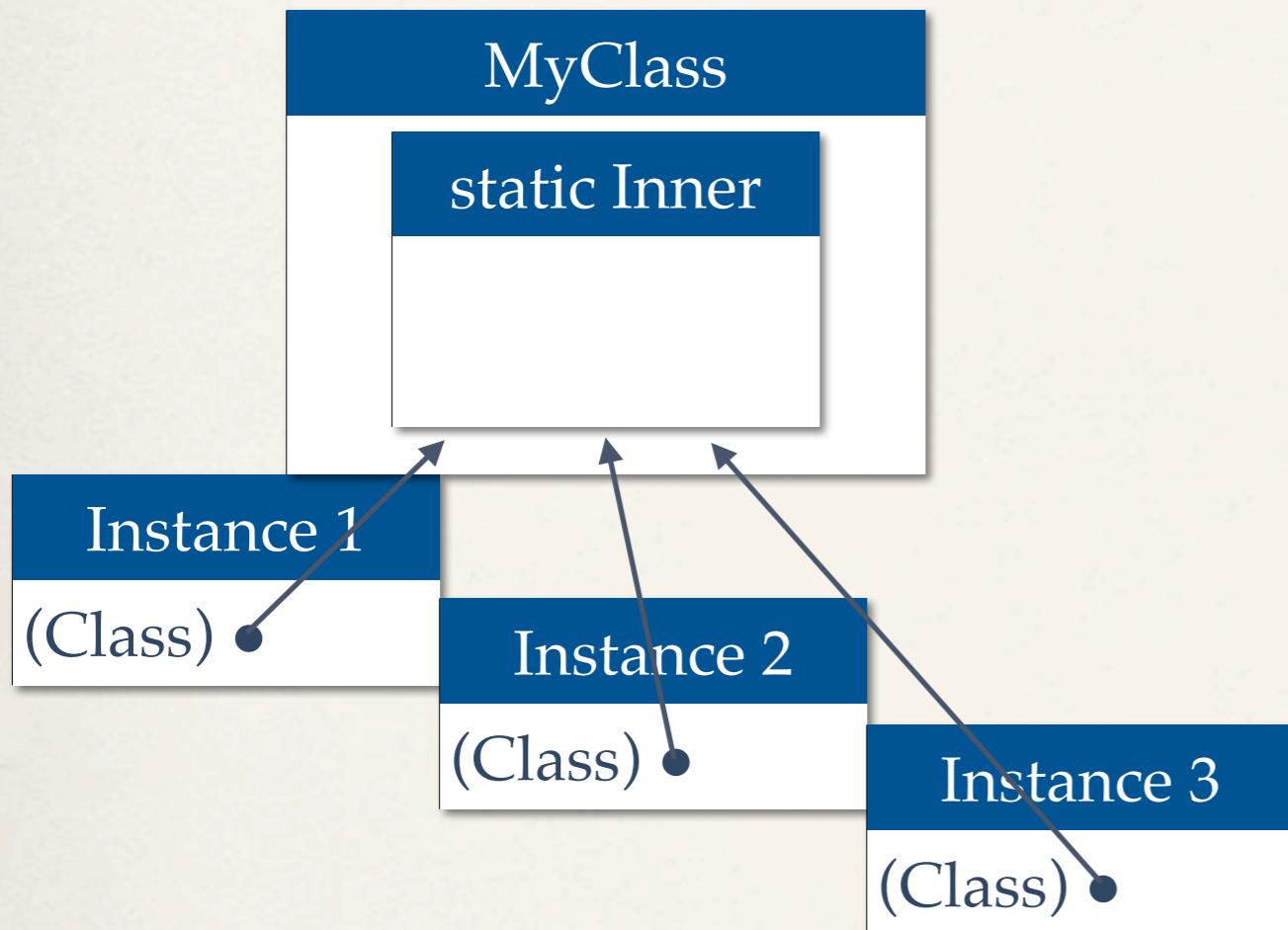
- オブジェクトの包含関係
- 外側のオブジェクトのメンバーの参照
- 構造的 this

# 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の参照
- 構造的 **this**

# Static vs Nonstatic Member Class

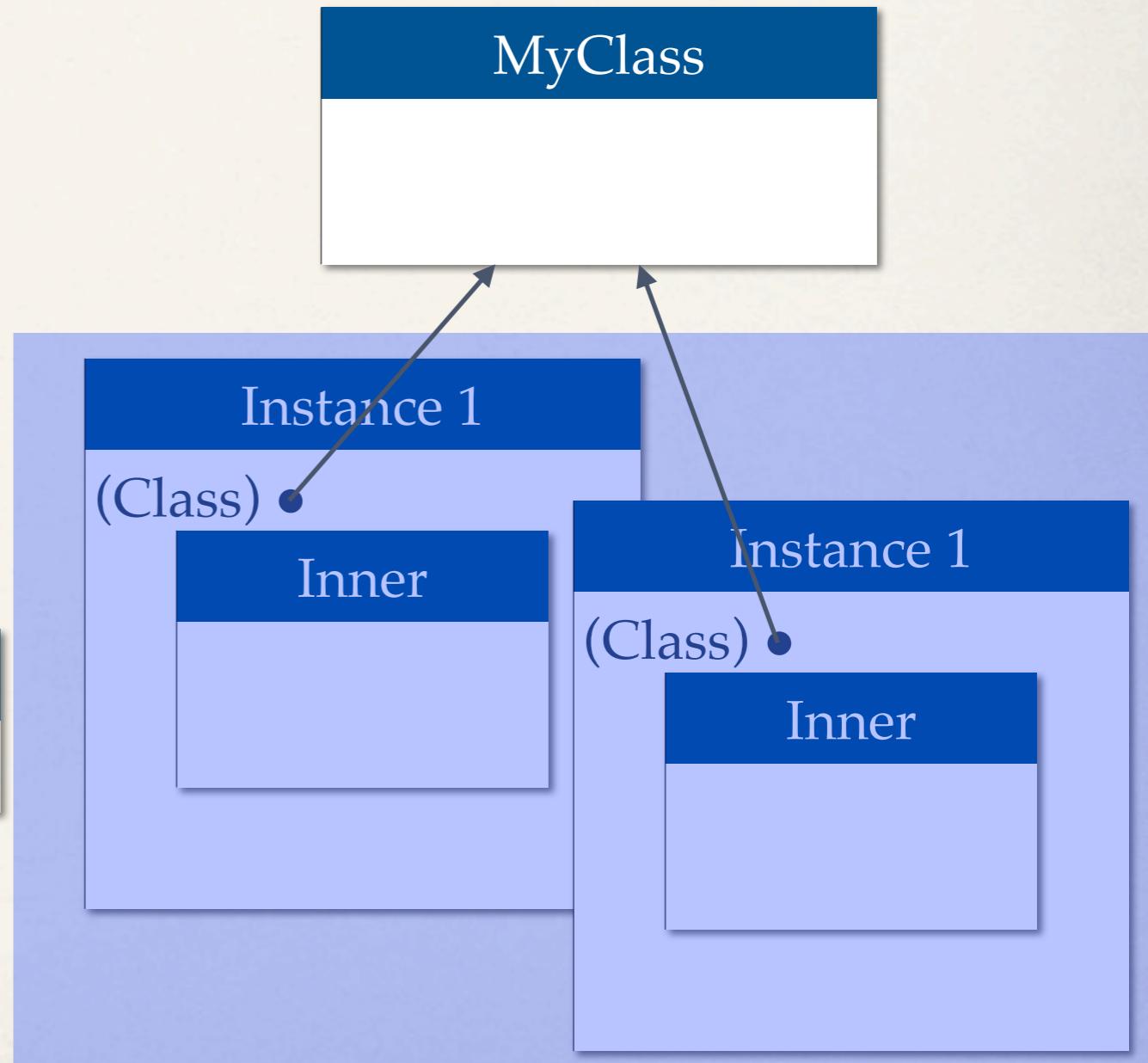
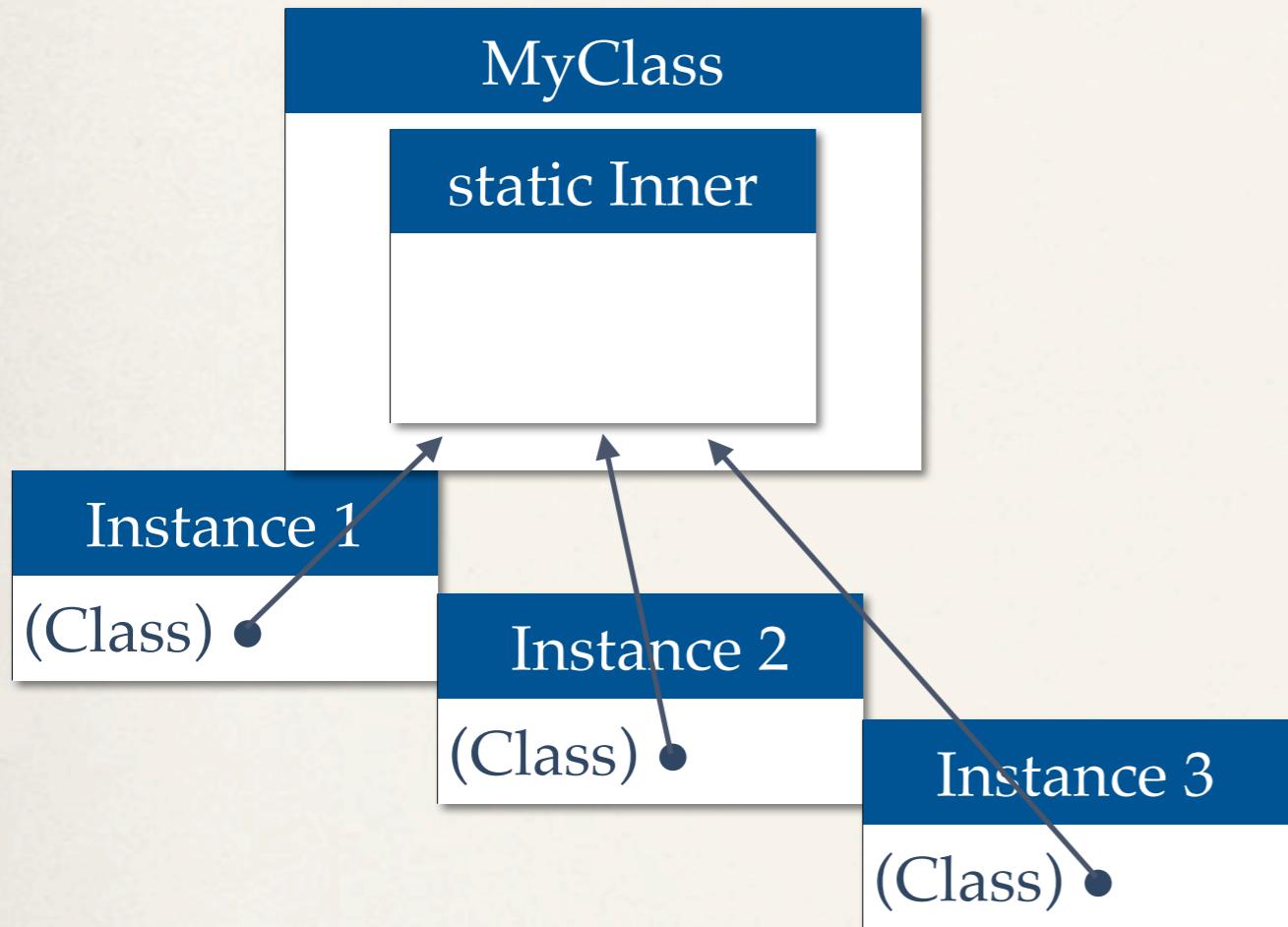


# Static

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。。。

# 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

```
import static java.lang.System.out;

public class AnonymousExample1 {
    private void run() {
        Object o1 = new Object();
        Object o2 = 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。"; }
        };
        Object o3;
        o3 = new Object() {
            public String toString() { return ""; }
        };
        for (Object o : new Object[] { o1, o2, o3 }) out.println("o = " + o);
    }

    public static void main(String[] _) {
        new AnonymousExample1().run();
    }
}
```

# Anonymous Class の例

```
import static java.lang.System.out;

public class AnonymousExample1 {
    private void run() {
        Object o1 = new Object();
        Object o2 = 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。"; }
        };
        Object o3;
        o3 = new Object() {
            public String toString() { return ""; }
        };
        for (Object o : new Object[] { o1, o2, o3 }) out.println("o = " + o);
    }

    public static void main(String[] _) {
        new AnonymousExample1().run();
    }
}
```

# Anonymous Class の例

```
import static java.lang.System.out;

public class AnonymousExample1 {
    private void run() {
        Object o1 = new Object();
        Object o2 = 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。"; }
        };
        Object o3;
        o3 = new Object() {
            public String toString() { return ""; }
        };
        for (Object o : new Object[] { o1, o2, o3 }) out.println("o = " + o);
    }

    public static void main(String[] _) {
        new AnonymousExample1().run();
    }
}
```

# Anonymous Class の例

---

o = java.lang.Object@7aab853b

o = 我輩はオブジェクトである。名前はまだない。

o =

# Anonymous class の用例

---

- ❖ Function object
- ❖ Process object

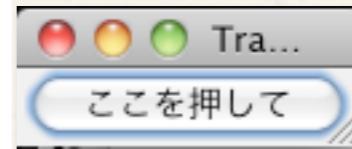
# Function Object

---

```
public class AnonymousExample2 extends JFrame {  
    private AnonymousExample2() {  
        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);  
    }  
  
    public static void main(String[] _) {  
        new AnonymousExample2();  
    }  
}
```

# Function Object

```
public class AnonymousExample2 extends JFrame {  
    private AnonymousExample2() {  
        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);  
    }  
  
    public static void main(String[] _) {  
        new AnonymousExample2();  
    }  
}
```



# Process Object

```
public class AnonymousExample2 {  
    private void run() {  
        new Thread() {  
            public void run() {  
                while (true) {  
                    out.println("こんにちは！");  
                    try { sleep(1000); } catch (InterruptedException e) {}  
                }  
            }  
        }.start();  
  
        new Thread() {  
            public void run() {  
                while (true) {  
                    out.println("お腹すいた！");  
                    try { sleep(2500); } catch (InterruptedException e) {}  
                }  
            }  
        }.start();  
    }  
  
    public static void main(String[] _) {  
        new AnonymousExample2().run();  
    }  
}
```

# 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

```
Problems @ Javadoc Declaration Console <terminated> AnonymousClassAsProcessExample [Java Application]
000002: パンが焼けたよ!
000003: パンを届けてきたよ!
000001: 生地をこね終ったよ!
001922: 生地をこね終ったよ!
002921: パンが焼けたよ!
003722: パンを届けてきたよ!
003823: 生地をこね終ったよ!
005724: 生地をこね終ったよ!
005823: パンが焼けたよ!
007423: パンを届けてきたよ!
007625: 生地をこね終ったよ!
008724: パンが焼けたよ!
009526: 生地をこね終ったよ!
```

# Process Object

```
public class AnonymousClassAsProcessExample extends JFrame {  
    private void run() {  
        final long start = System.currentTimeMillis();  
  
        new Thread() {  
            public void run() {  
                while (true) {  
                    long t = System.currentTimeMillis() - start;  
                    out.printf("%06d: 生地をこね終ったよ！\n", t);  
                    try { sleep(1900); } 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
- (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
- \* (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
- \* (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
- ✳ (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

# プログラム (11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