

計算機科學第一

2012年度第7回

脇田建

もくじ

- ♣10: Always override `toString`
- ♣13: Minimize accessibility
- ♣14: Public Class

X: Always override `toString`

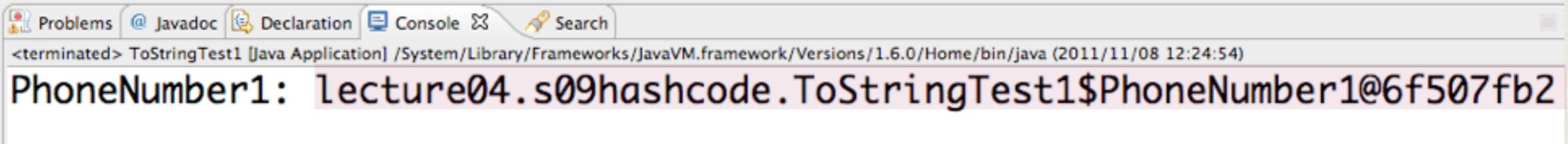
普通 `toString` は

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 the Eclipse IDE interface with the 'Console' tab selected. The console window displays the output of the Java application 'ToStringTest1'. The output shows the class name and its memory address: 'PhoneNumber1: lecture04.s09hashcode.ToStringTest1\$PhoneNumber1@6f507fb2'. The rest of the console window is empty.

```
Problems @ Javadoc Declaration Console Search  
<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 an IDE interface with a toolbar at the top featuring 'Problems', 'Javadoc', 'Declaration', 'Console', and 'Search' buttons. The 'Console' tab is active. Below the toolbar, the console output window displays:

```
<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 shows two instances of the PhoneNumber class: 'PhoneNumber1' and 'PhoneNumber2'. 'PhoneNumber1' is represented by its memory address, while 'PhoneNumber2' is correctly formatted as '(03)5734-3493'.

toString を上書きすると

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

The screenshot shows an IDE interface with a toolbar at the top and a console window below. The toolbar includes icons for Problems, Javadoc, Declaration, Console, and Search. The console window displays the output of a Java application named 'ToStringTest1'. The output shows two instances of the class: 'PhoneNumber1' and 'PhoneNumber2'. 'PhoneNumber1' is printed as a long string representing its formatted value, while 'PhoneNumber2' is printed as its memory address.

```
Problems @ Javadoc Declaration Console Search
<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
```

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 the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following text:

```
Problems Javadoc Declaration Console Search  
<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
```

The output shows that the custom toString method for PhoneNumber1 is being used, while the default toString method for PhoneNumber2 is used. The hashCode value for phone1 is also displayed.

toString を上書きしたらやつておきたい こと

- `toString` の出力形式を使った Constructor
- `toString` に関す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("\\(((\\d)+)\\)(\\d)+)-((\\d)+)");  
    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 のアクセサ

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

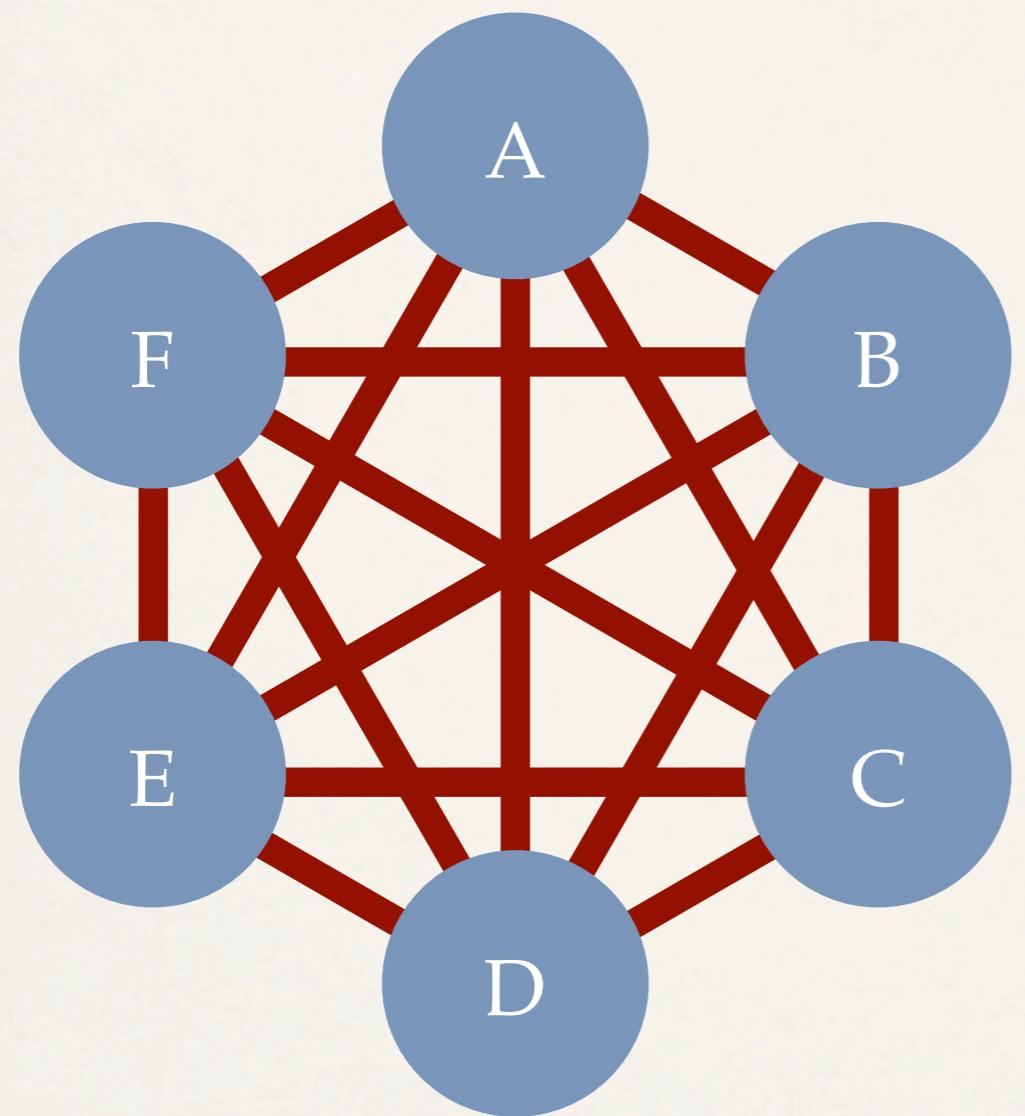
13 Minimize Accessibility

情報隱蔽

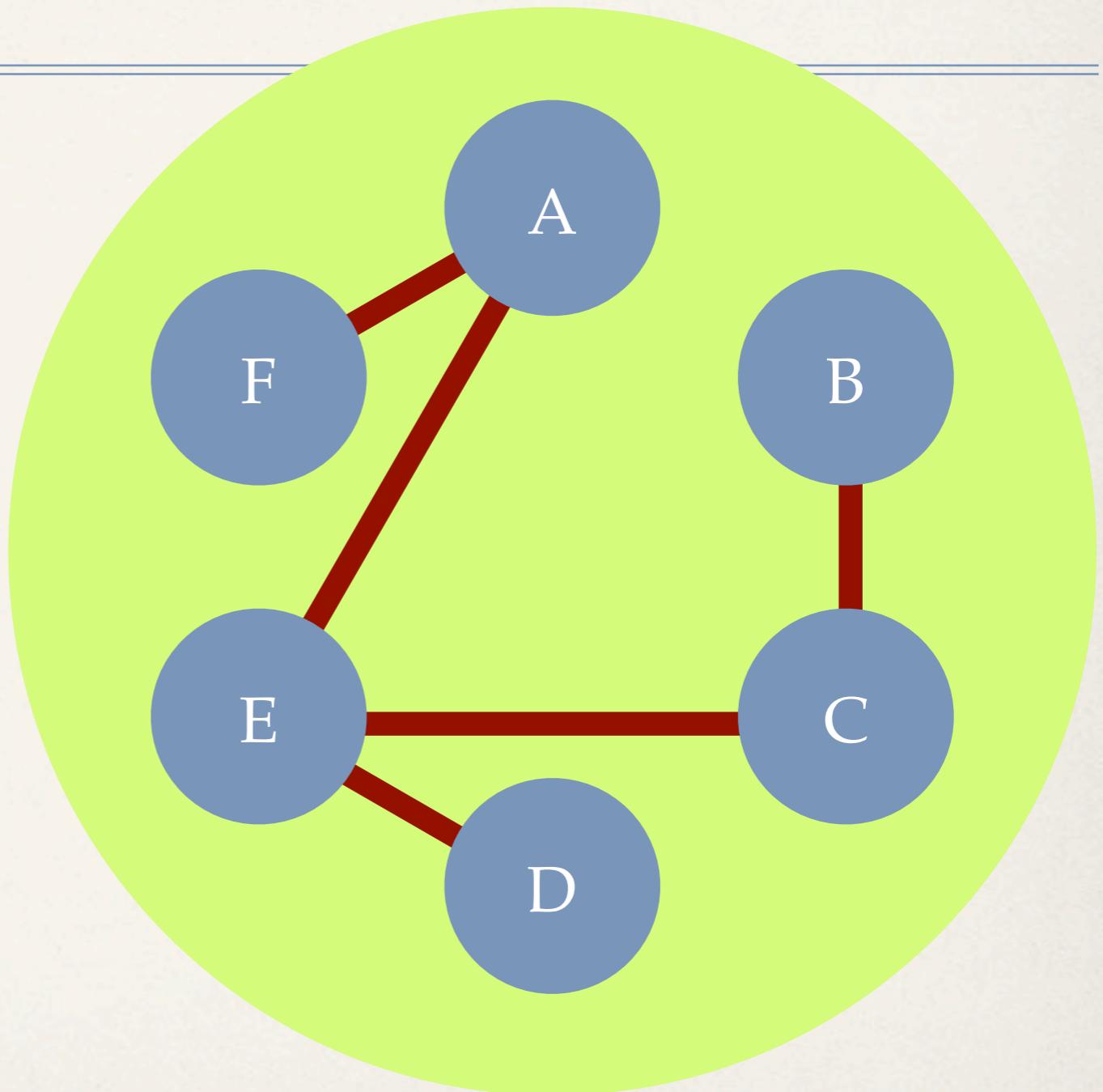
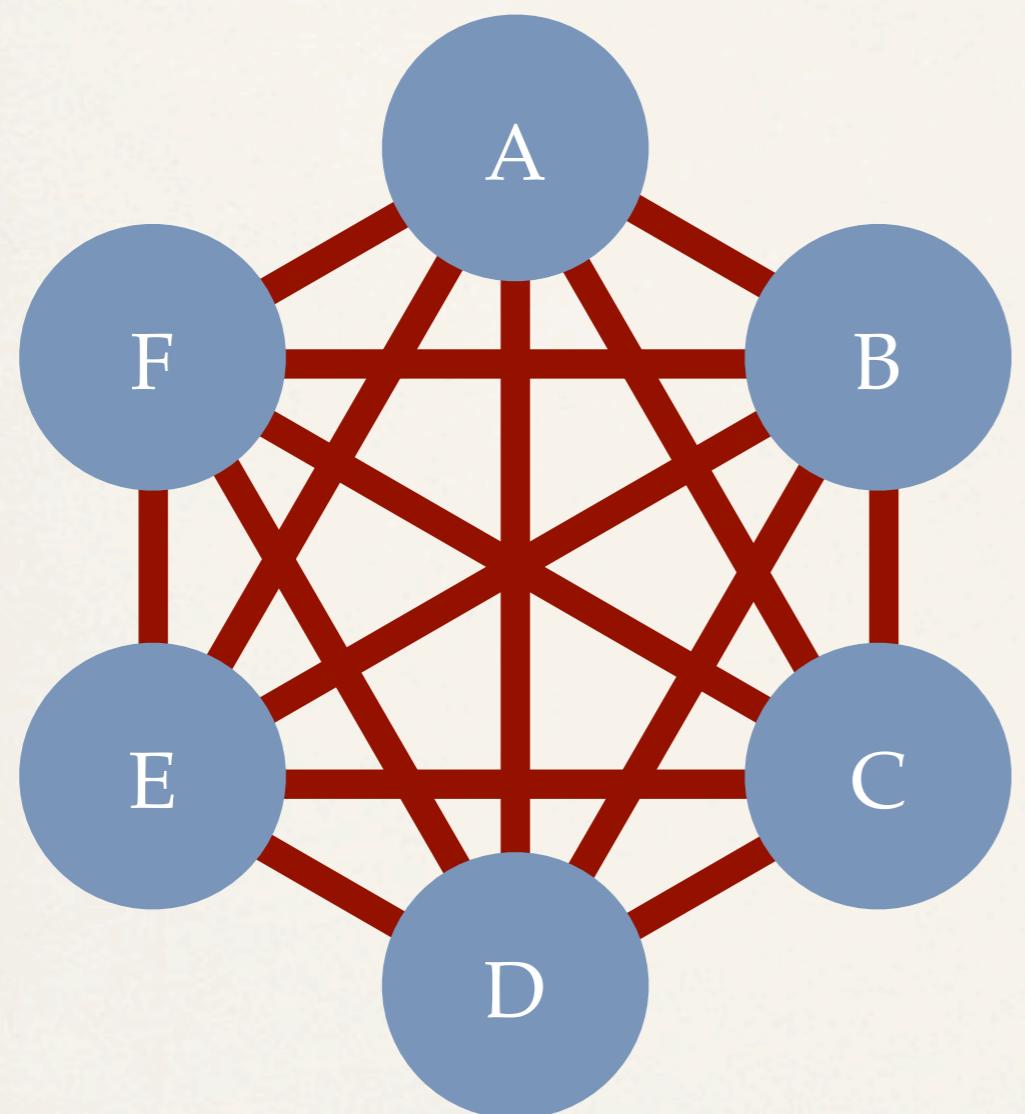
情報隠蔽の効用

1. モジュールの分離 (decouple) → 独立性の向上
2. 開発の効率化 → 並行開発
3. 保守労力の軽減
4. 性能上の問題の診断が容易
5. ソフトウェア再利用性の向上
6. コンパクトなシステムの開発

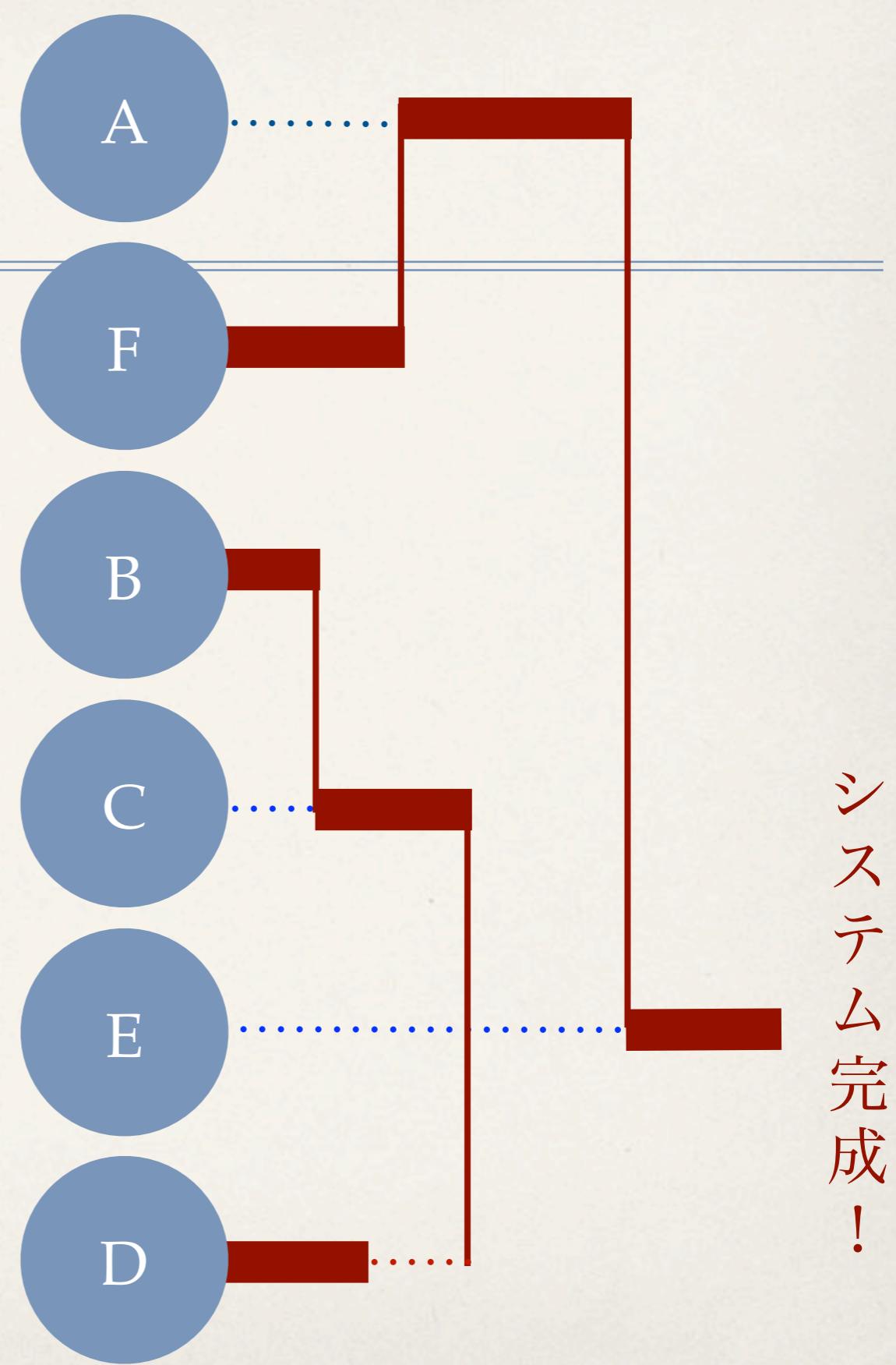
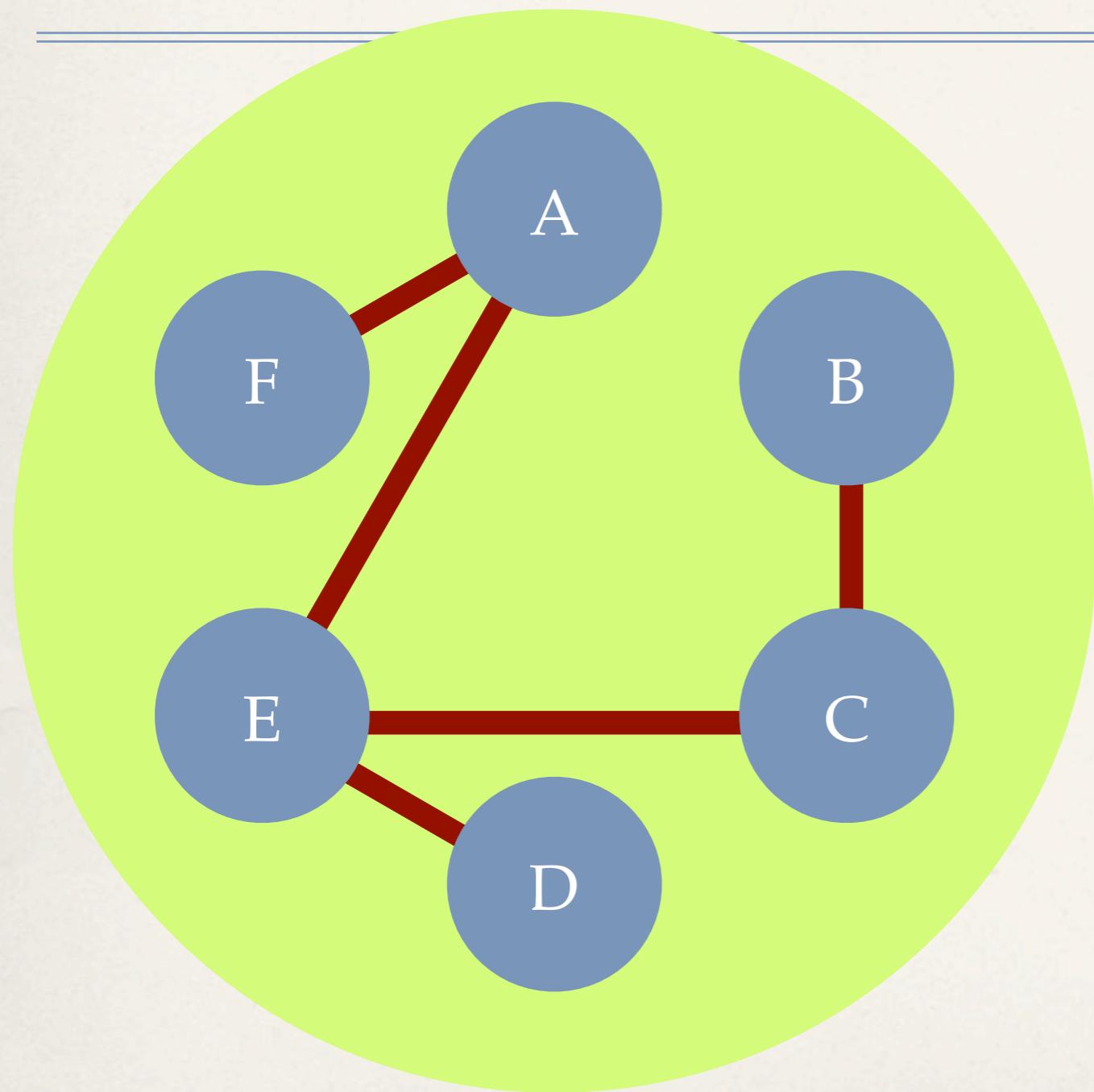
(1) 独立性の向上



(1) 独立性の向上

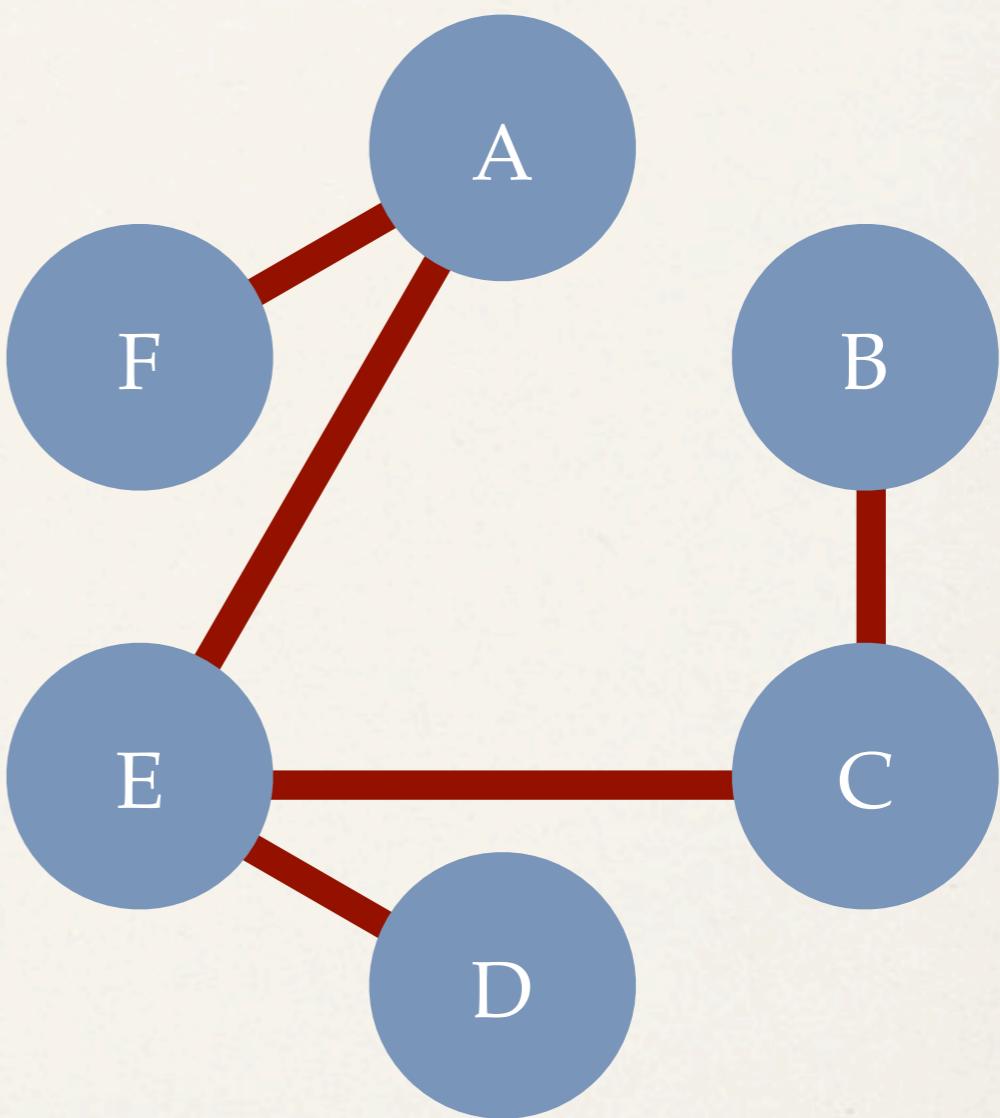
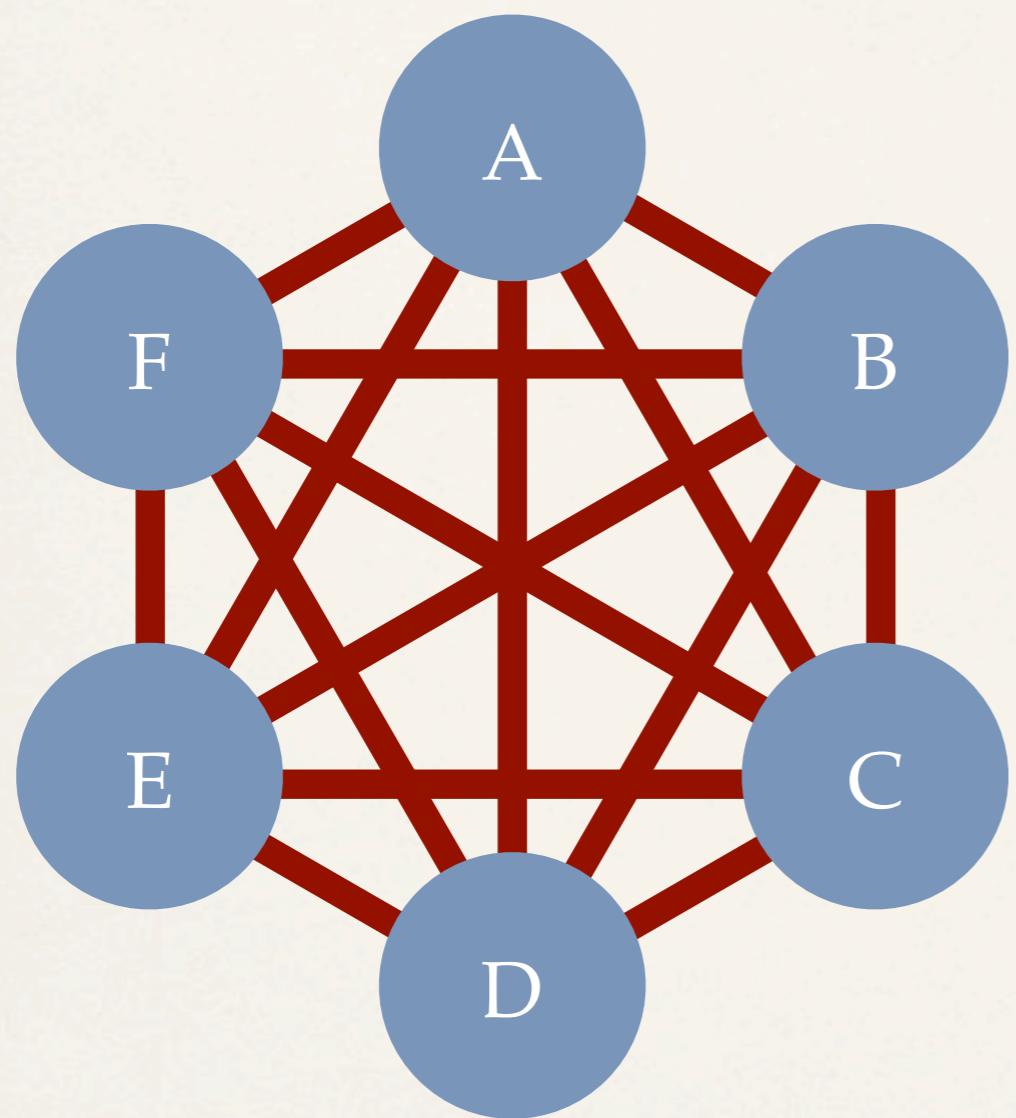


(2) 並行開発

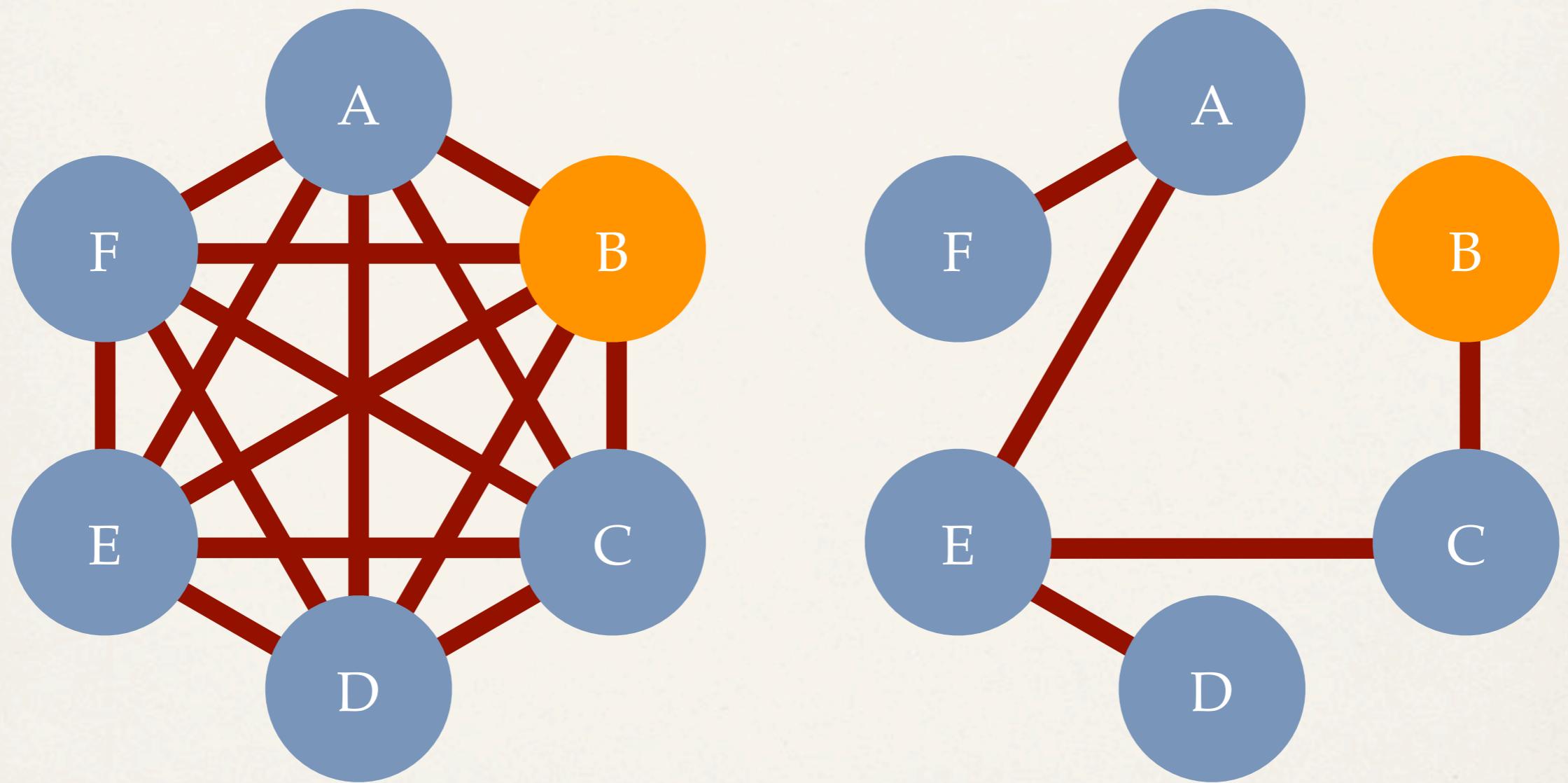


システム完成！

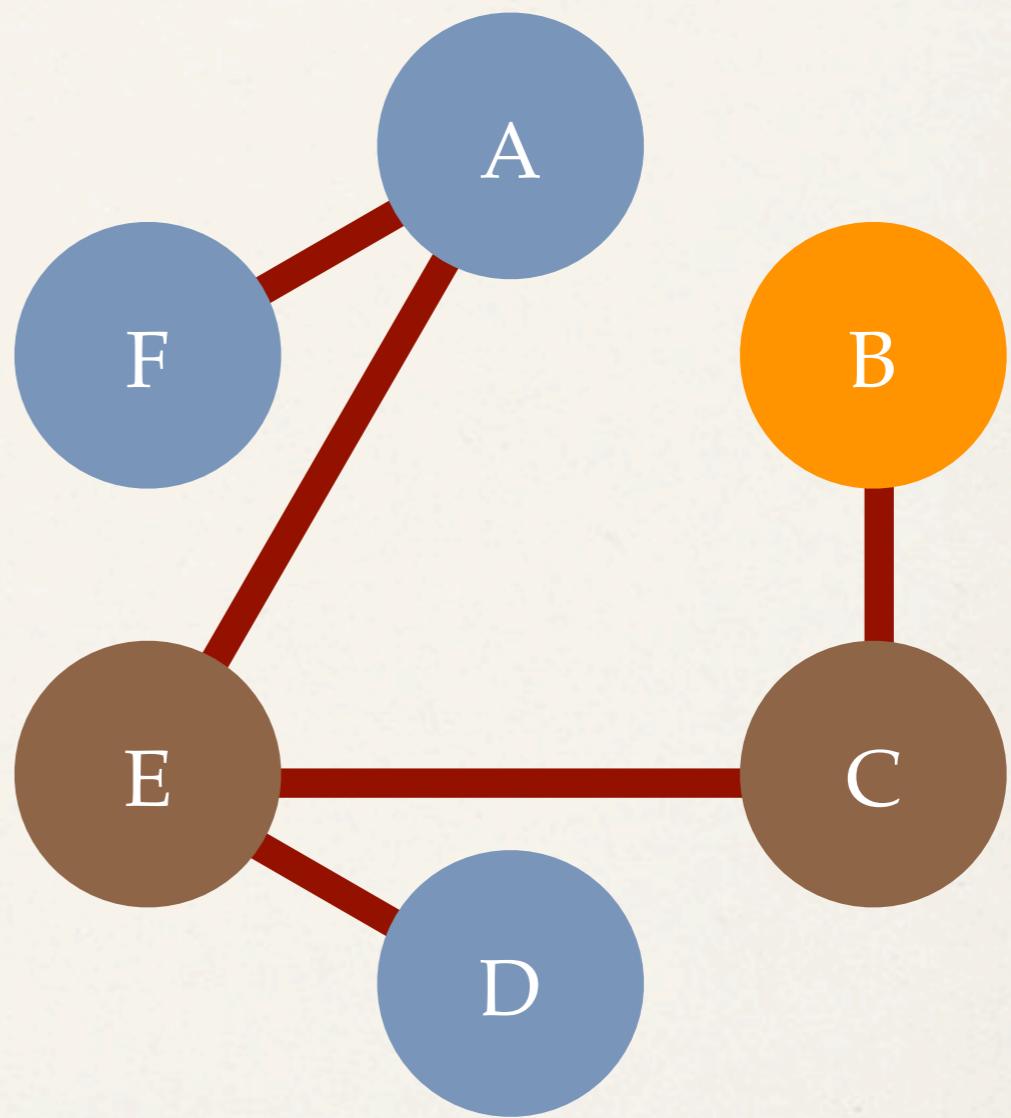
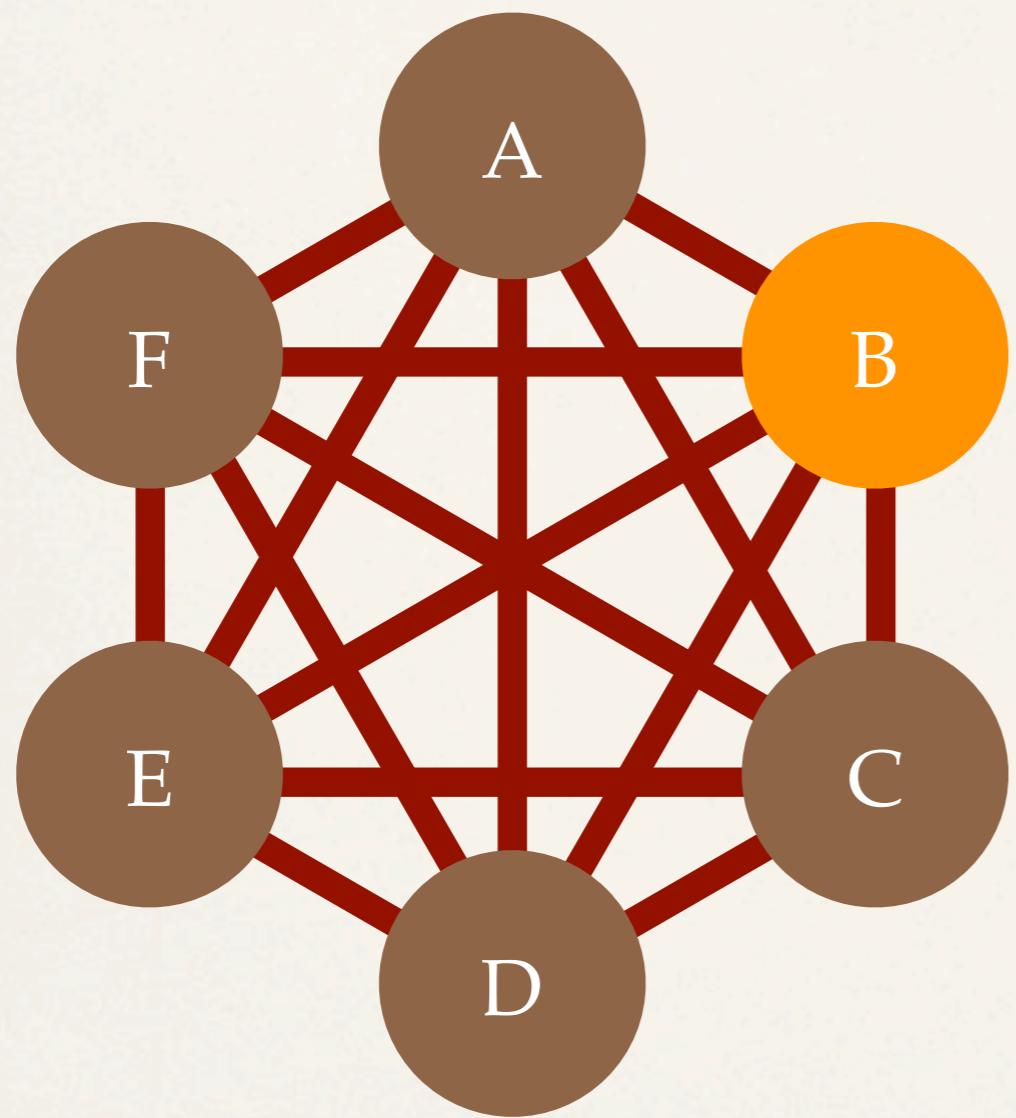
(3) 保守劳力



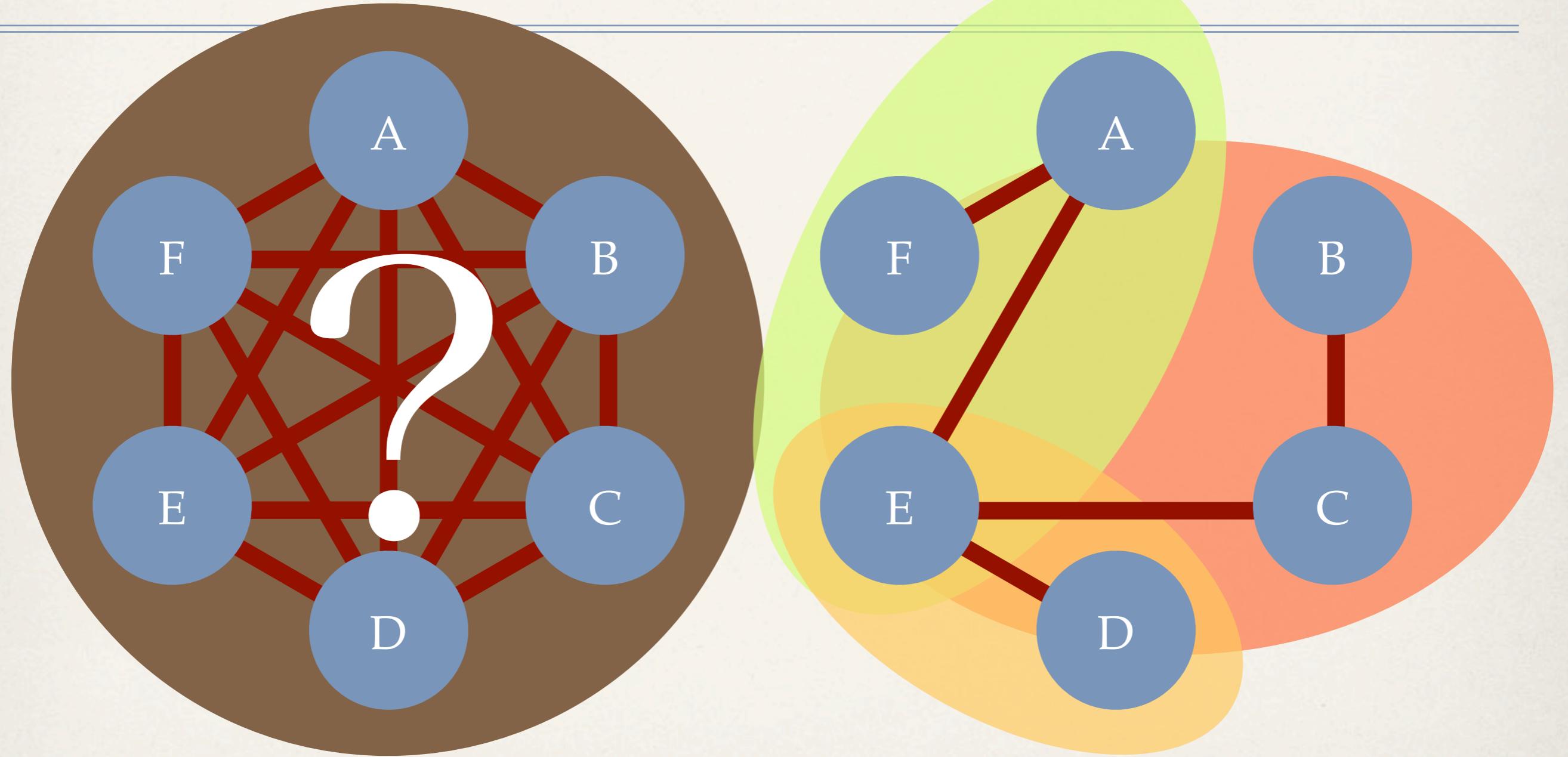
(3) 保守労力:仕様変更！



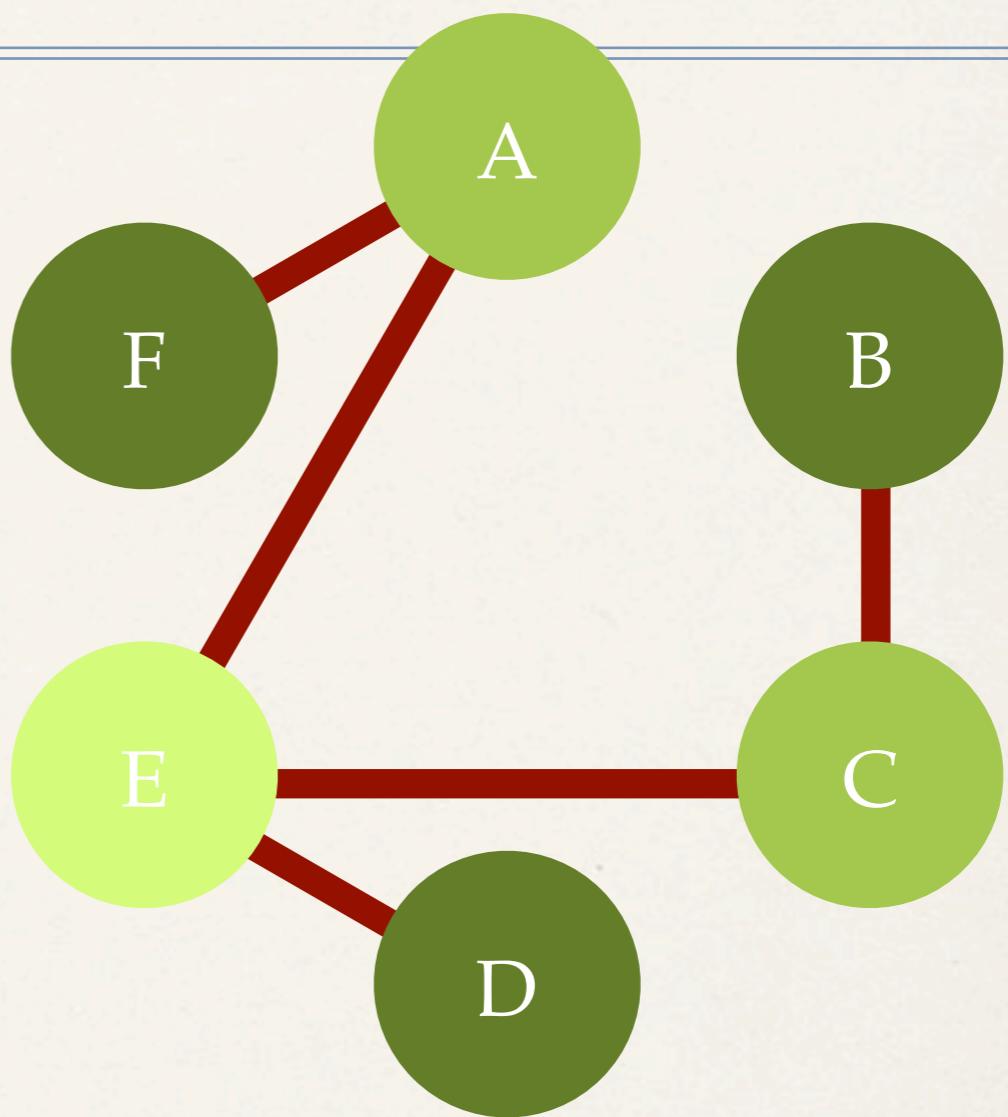
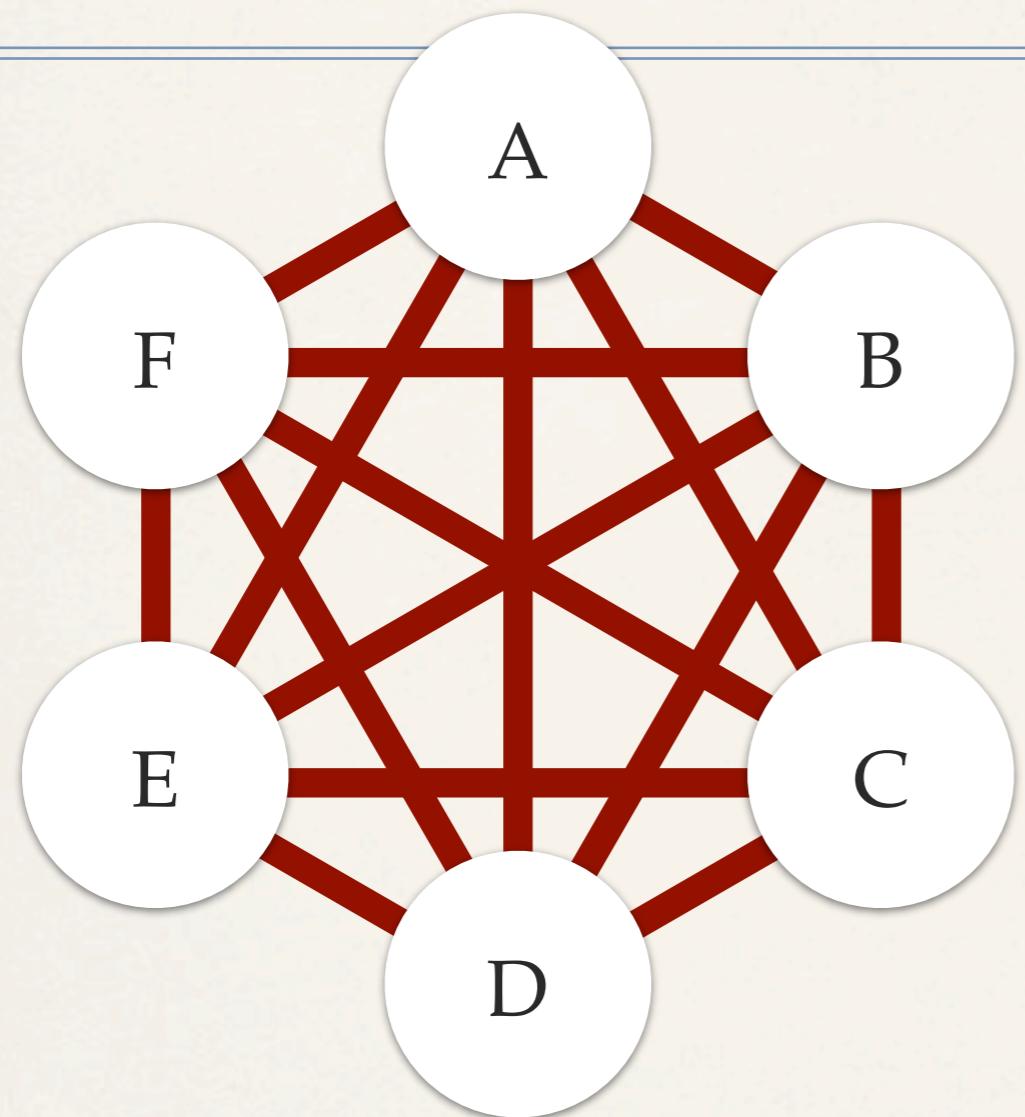
(3) 保守労力:仕様変更！



(4) 性能上の問題の診断



(5) 再利用性



?

悪

良

優

(6) コンパクト

- * 単体テスト(モジュール単位でのテスト)が実施できれば、システムが未完の状態でも不安にかられて不必要的機能を追加しないですむ。

Java と 情報隠蔽

- スコープ規則
- アクセス制御 (private / protected / public)
- 基本 → 可能な限りアクセス **できなく** する
- アクセス手段が少ない → このクラスを気にかける心配が少ない

アクセス制御

アクセスを許されるクラス

private

宣言したクラス内のみ

(package-private)

同じパッケージ内のクラス
(defaultアクセス)

protected

(a) サブクラス
(b) 同じパッケージ内のクラス

public

すべてのクラス

大原則

1. オブジェクトを指すフィールド

→ どんなことがあってもpublicはだめ

* publicで代入可能なフィールド → 並列化不可能

2. public static final 配列 → セキュリティホール

3. 配列を返すアクセサ → セキュリティホール

public objectの危険性

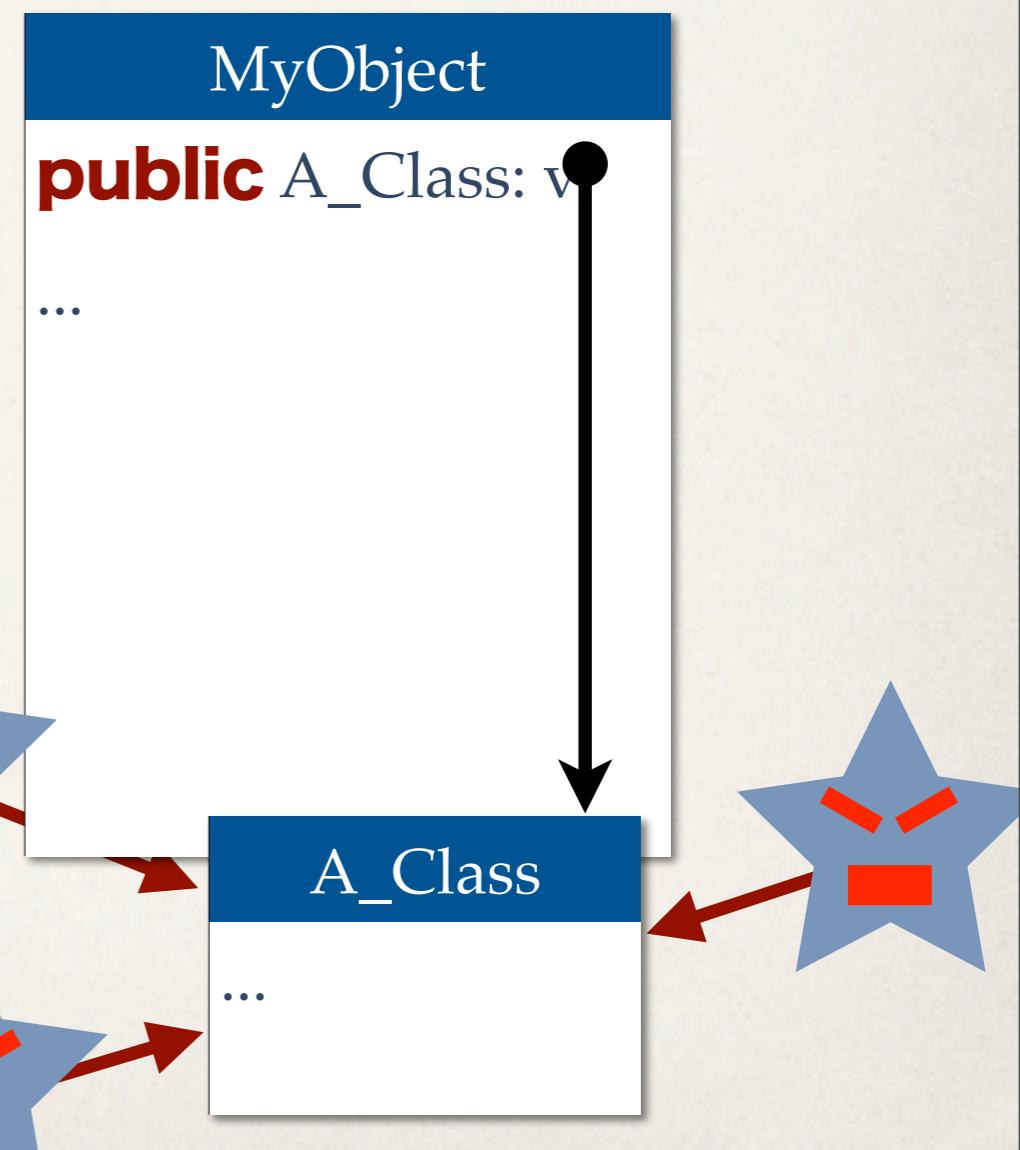
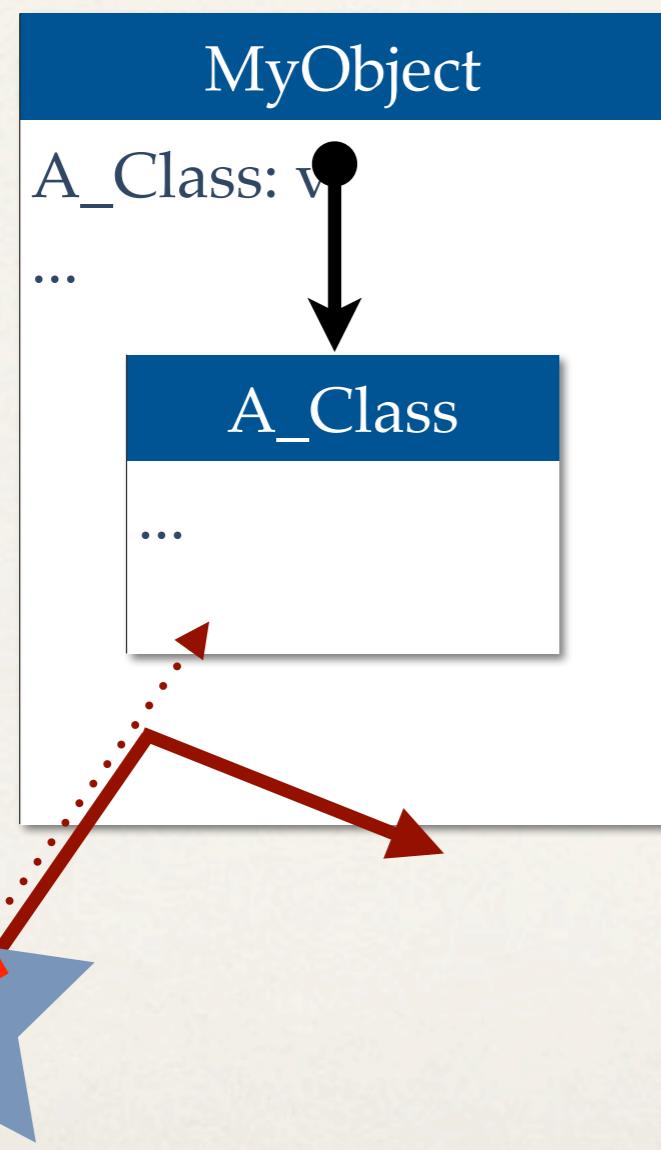
MyObject

```
A_Class: v;  
...
```

MyObject

```
public A_Class: v;  
...
```

public objectの危険性



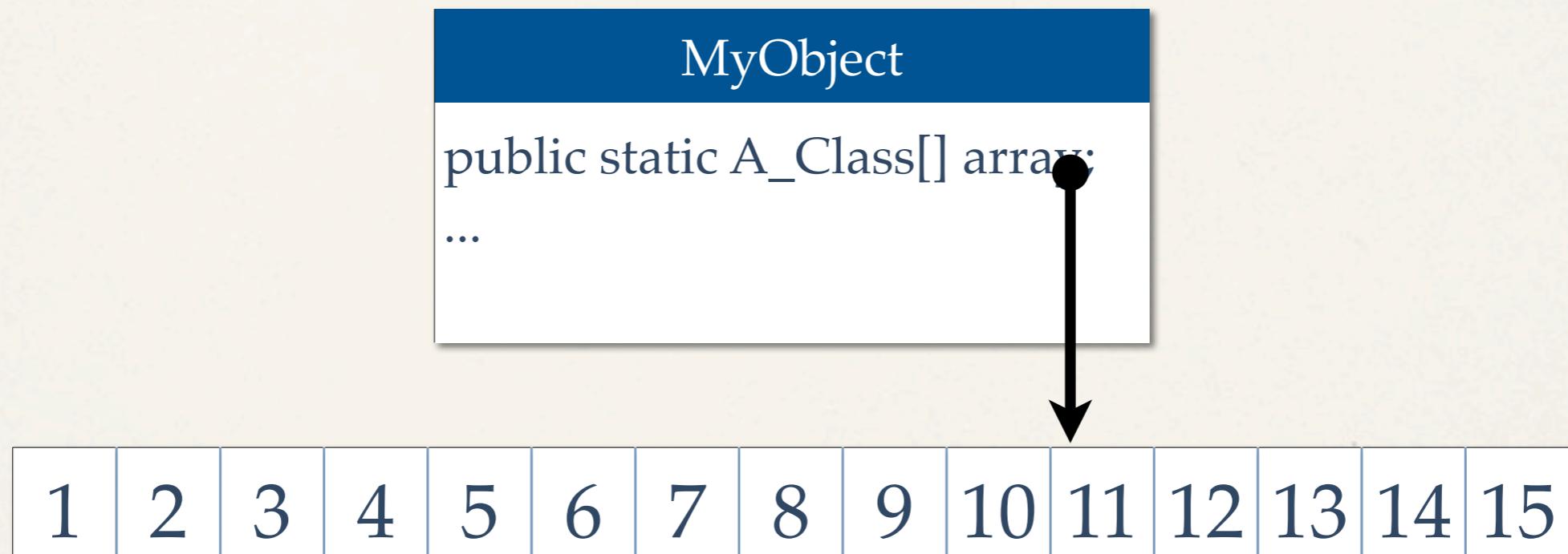
public static 配列

MyObject

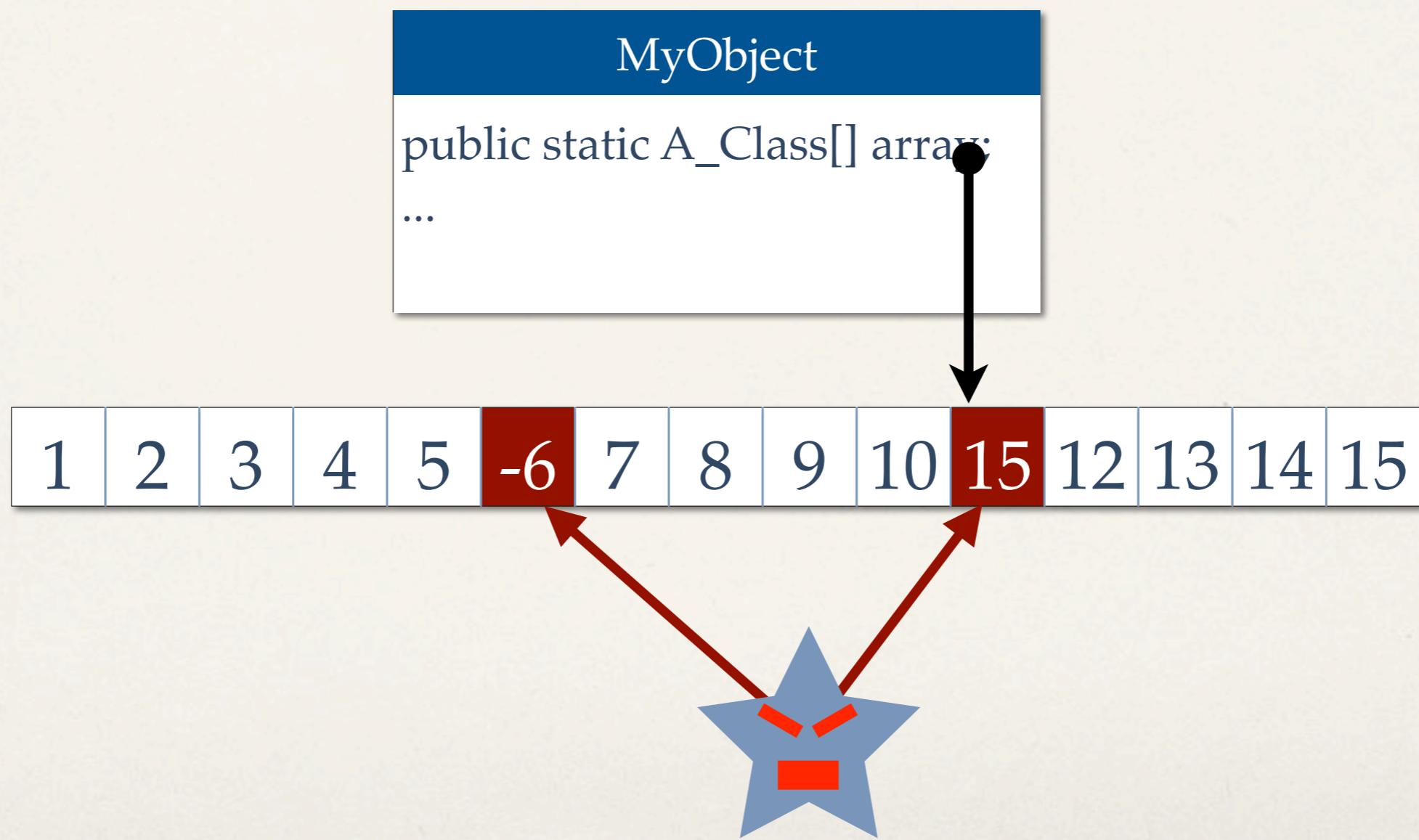
```
public static A_Class[] array;
```

...

public static 配列



public static 配列

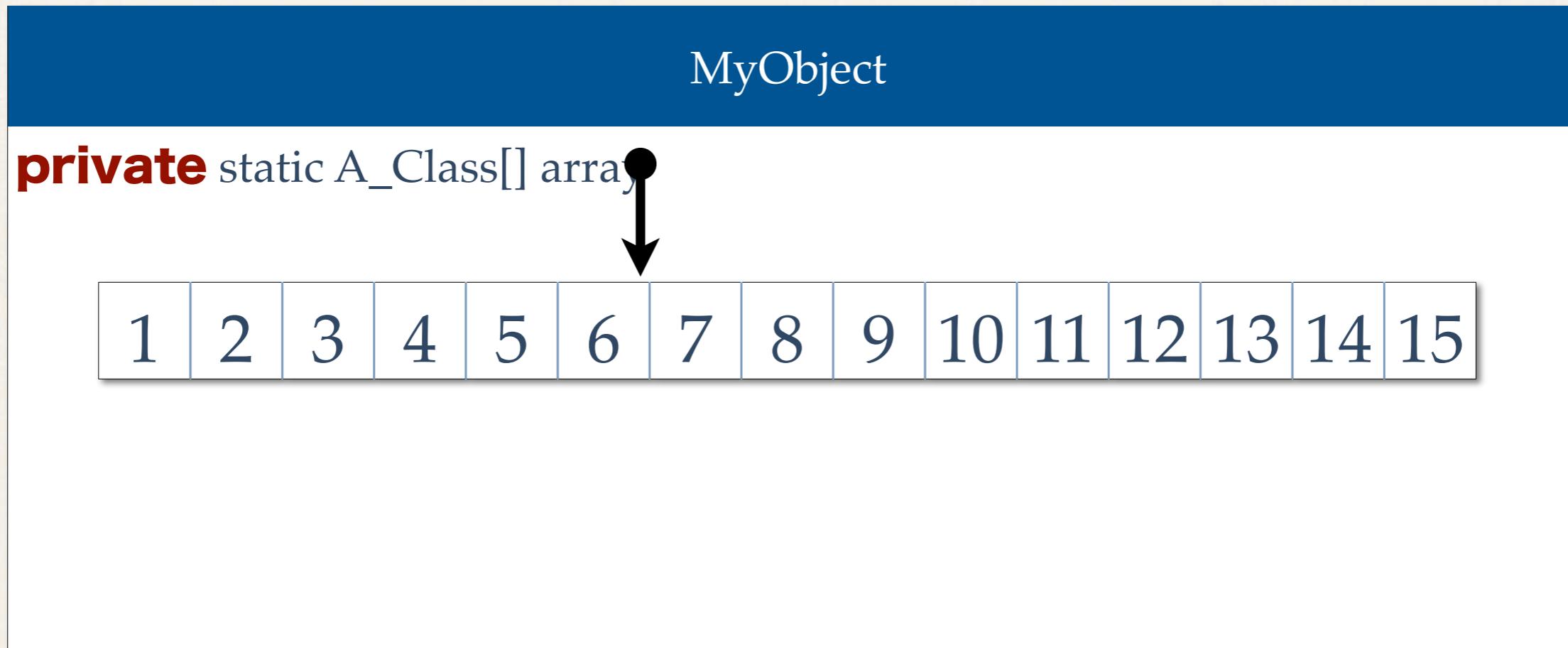


配列を返すaccessor

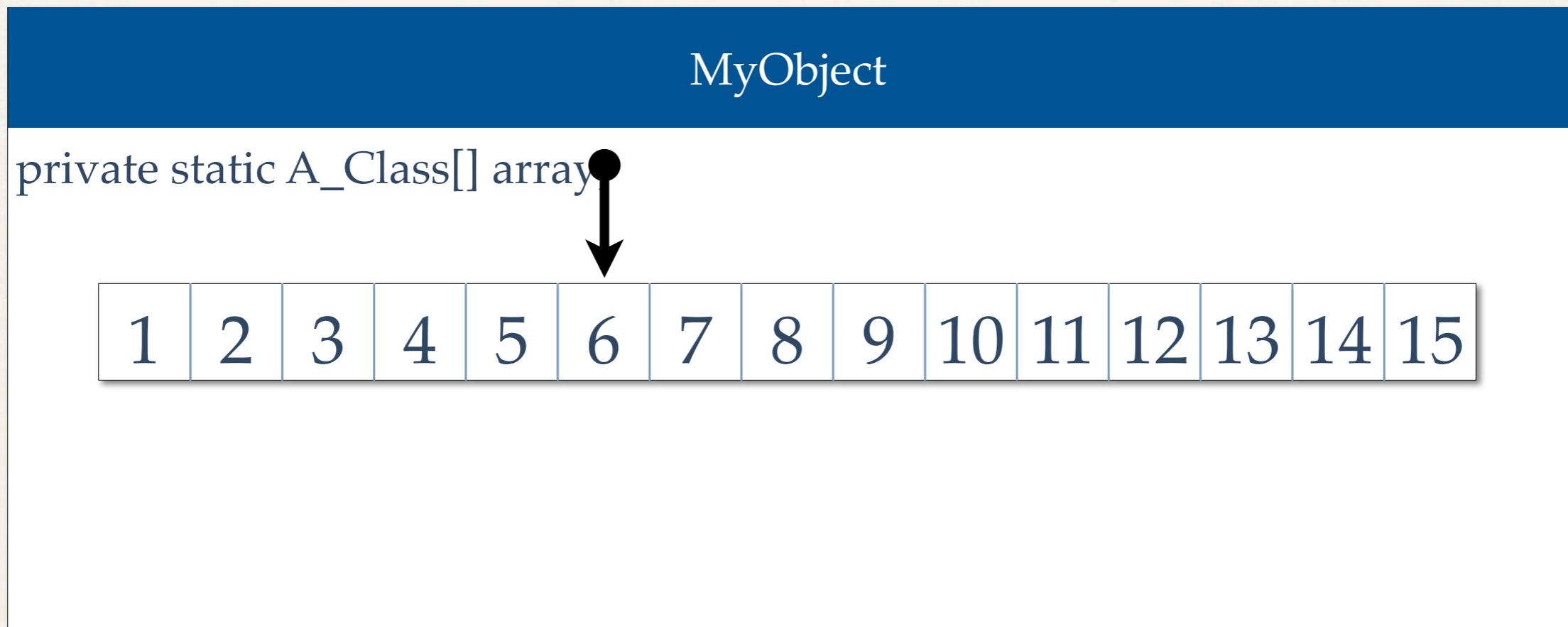
MyObject

private static A_Class[] array;

配列を返すaccessor



配列を返すaccessor



でも、外から配列にアクセスしたいこともある。。。。

配列を返すaccessor

MyObject

private static A_Class[] array



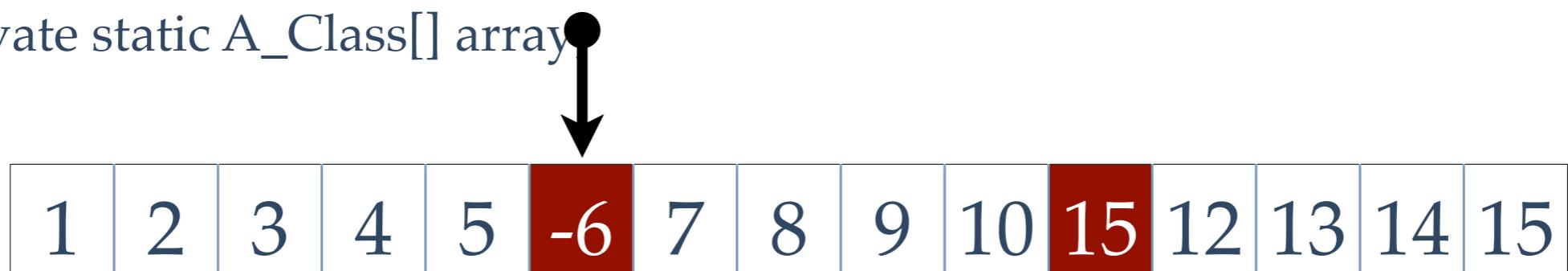
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

public A_Class[] getArray() { return array; } // 配列の accessor

配列を返すaccessor

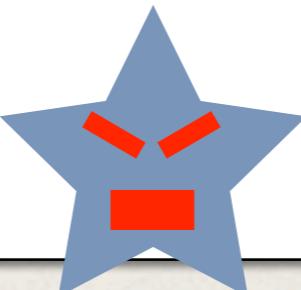
MyObject

private static A_Class[] array



public A_Class[] getArray() { return array; } // 配列の accessor

```
aMethod() {
    A_Class[] array = myObject.getArray();
    array[6] = -6;
    array[11] = 15;
}
```



正しいaccessor(1)

MyObject

```
private static A_Class[] array
```

A black arrow points downwards from the variable 'array' to the center of a 15-element array. The array is represented by a row of 15 boxes, each containing a number from 1 to 15. The boxes are separated by thin vertical lines.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

```
public A_Class getElementOfArray(int i) { return array[i]; }
```

```
aMethod() {  
    A_Class elem = myObject.getElementOfArray(6);  
    elem.doSomething();  
}
```



正しい accessor (2)

MyObject

```
private static A_Class[] array
```

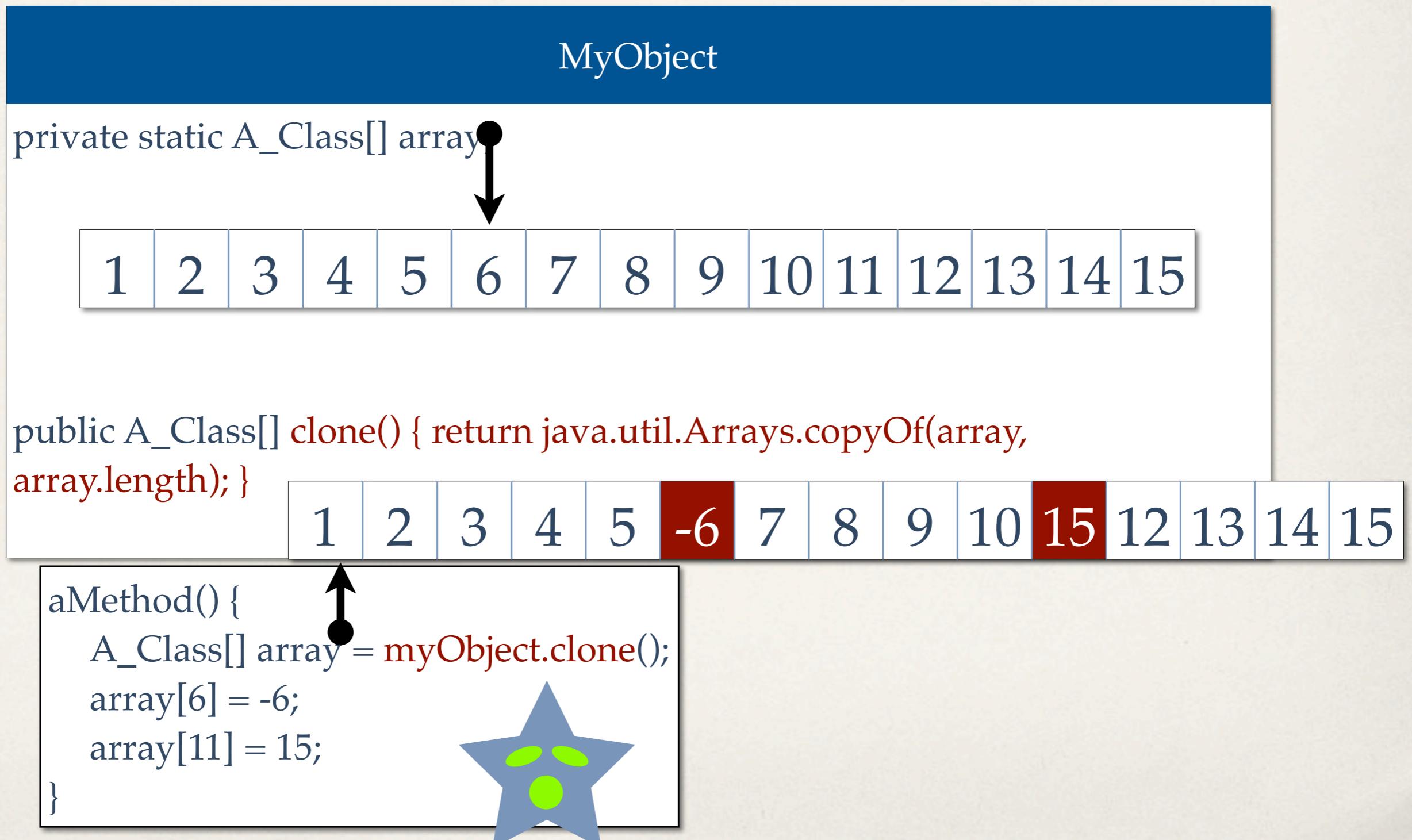
A diagram showing a downward arrow pointing from the variable 'array' to a horizontal row of 15 boxes, each containing a number from 1 to 15. The boxes are separated by vertical lines.

```
public A_Class[] clone() { return java.util.Arrays.copyOf(array, array.length); }
```

```
aMethod() {  
    A_Class[] array = myObject.clone();  
    array[6] = -6;  
    array[11] = 15;  
}
```



正しい accessor (2)



正しいaccessor (3)

MyObject

```
private static A_Class[] private_values;  
  
public static final List<A_Class> values =  
  
    Collections.unmodifiableList(Arrays.asList(private_values))  
);
```

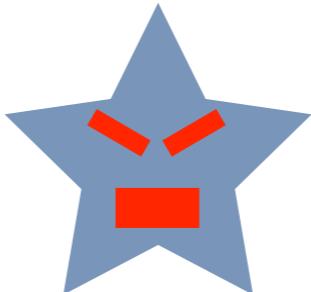
正しいaccessor (3)

MyObject

```
private static A_Class[] private_values;  
  
public static final List<A_Class> values =  
  
    Collections.unmodifiableList(Arrays.asList(private_values))  
);
```

Unsupported
Operation
Exception

```
aMethod() {  
    List<A_Class> values = myObject.values;  
    values.get(6);  
    values.set(6, -6);  
    values.set(11, 15);  
}
```



14 Public Class

公開されたクラスの

~~public フィールド~~

日付

(x, y)-座標を表したい

二つの流儀

Point

```
public x;  
public y;
```

Point

```
private x;  
private y;
```

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

```
public getX() { return x; }  
public getY() { return y; }
```

```
public setX(x) { this.x = x; }  
public setY(y) { this.y = y; }
```

二つの流儀

Point

```
public x;  
public y;
```

面倒？
いや、そうでもない。
Refactoring がある！

Point

```
private x;  
private y;
```

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

```
public getX() { return x; }  
public getY() { return y; }
```

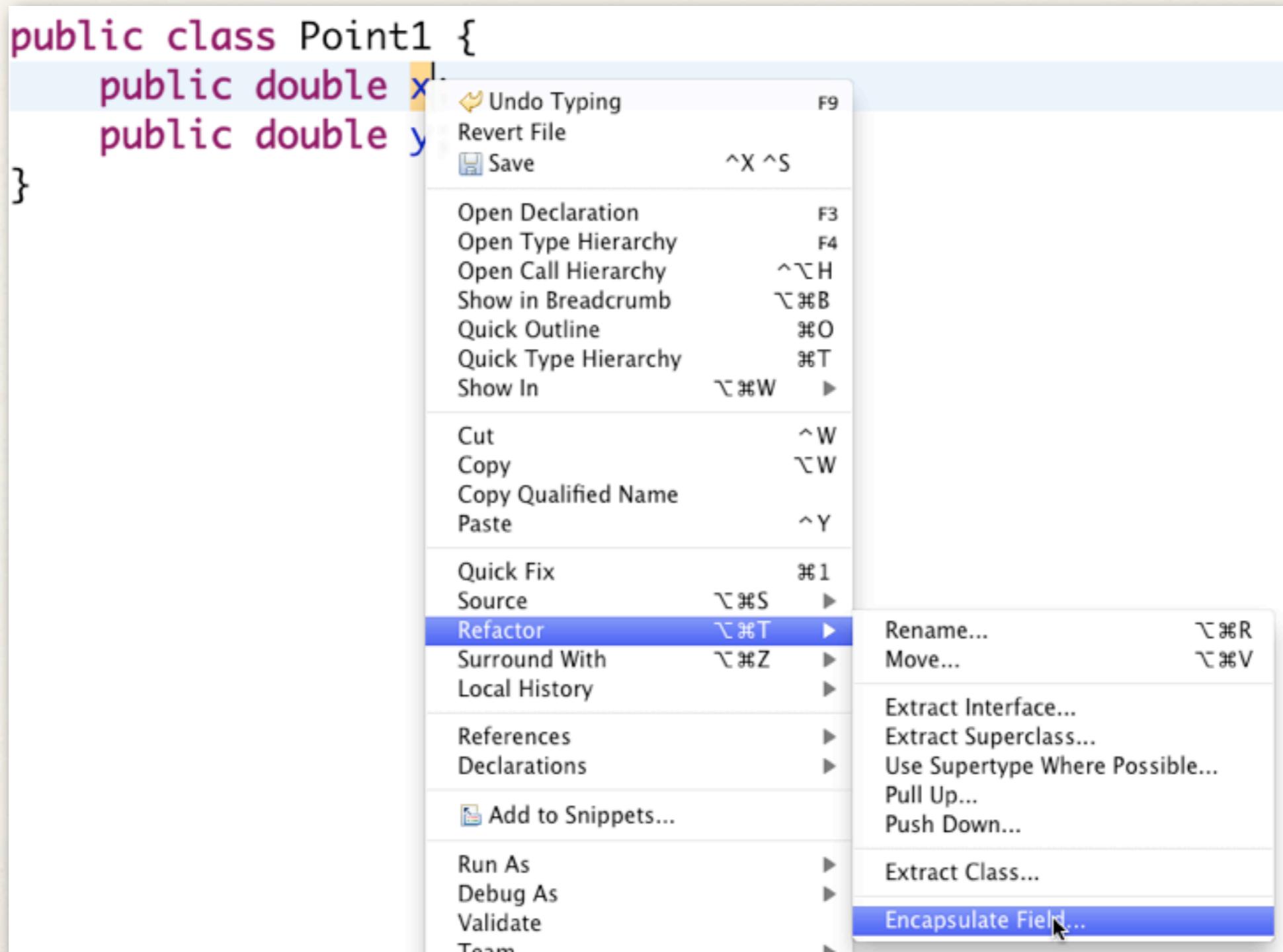
```
public setX(x) { this.x = x; }  
public setY(y) { this.y = y; }
```

Refactoring (1/4)

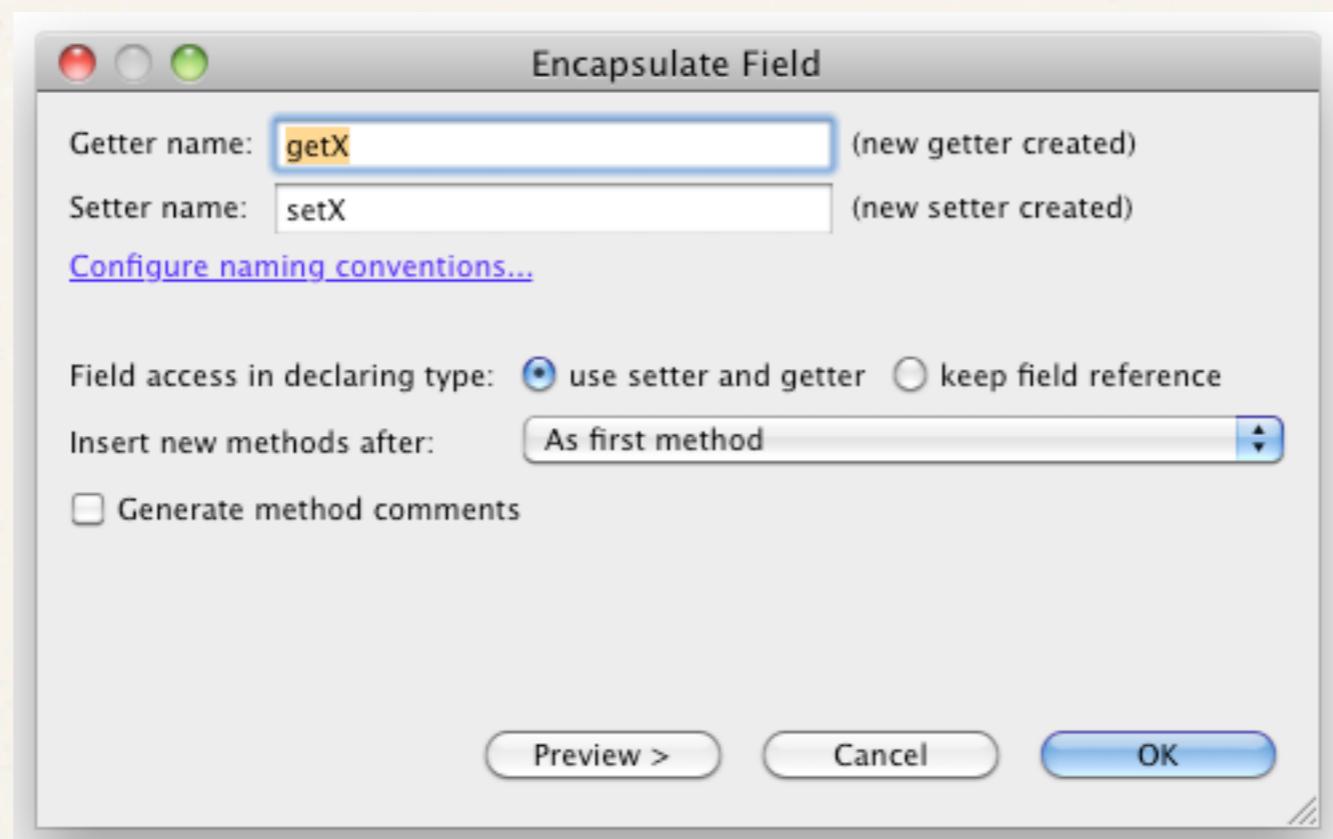
```
package lecture04.s14public_classes;

public class Point1 {
    public double x;
    public double y;
}
```

Refactoring (2/4)



Refactoring (3/4)



Refactoring (4/4)

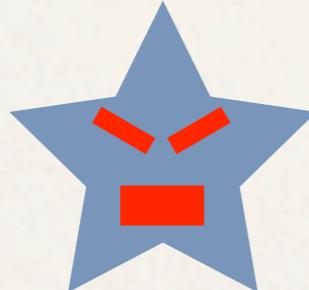
```
package lecture04.s14public_classes;

public class Point1 {
    private double x;
    public double y;
    public void setX(double x) {
        this.x = x;
    }
    public double getX() {
        return x;
    }
}
```

二つの流儀

Point

```
public x;  
public y;
```



Point

```
private x;  
private y;
```

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

```
public getX() { return x; }  
public getY() { return y; }
```

```
public setX(x) { this.x = x; }  
public setY(y) { this.y = y; }
```

二つの流儀

Point

```
public x;  
public y;
```

判断の分かれ目

Public class or not public?

Point

```
private x;  
private y;
```

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

```
public getX() { return x; }  
public getY() { return y; }
```

```
public setX(x) { this.x = x; }  
public setY(y) { this.y = y; }
```

二つの流儀

**private/package private
class なら OK**

Point

```
public x;  
public y;
```

判断の分かれ目

Public class or not public?

public class ならこっち

Point

```
private x;  
private y;
```

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

```
public getX() { return x; }  
public getY() { return y; }
```

```
public setX(x) { this.x = x; }  
public setY(y) { this.y = y; }
```

Java標準APIでの失敗例

java.awt.Dimension

```
public int width;  
public int height;
```

java.awt.Component

```
private int width;  
private int height;  
  
public Dimension getSize() {  
    return new Dimension(this.width,  
                        this.height);  
}
```

こうあるべきだった(1/2)

java.awt.Dimension

```
public final int width;  
public final int height;
```

java.awt.Component

```
private Dimension size =  
    new Dimension(width, height);
```

```
public Dimension getSize() { return size; }
```

こうあるべきだった(2/2)

java.awt.Dimension

```
public int width;  
public int height;
```

java.awt.Component

```
private int width;  
private int height;  
  
public Dimension getSize(){  
    return new Dimension(this.width,  
                        this.height);  
}  
  
public int getWidth() { return this.width; }  
public int getHeight() { return this.height; }  
  
Java 1.2 で追加
```