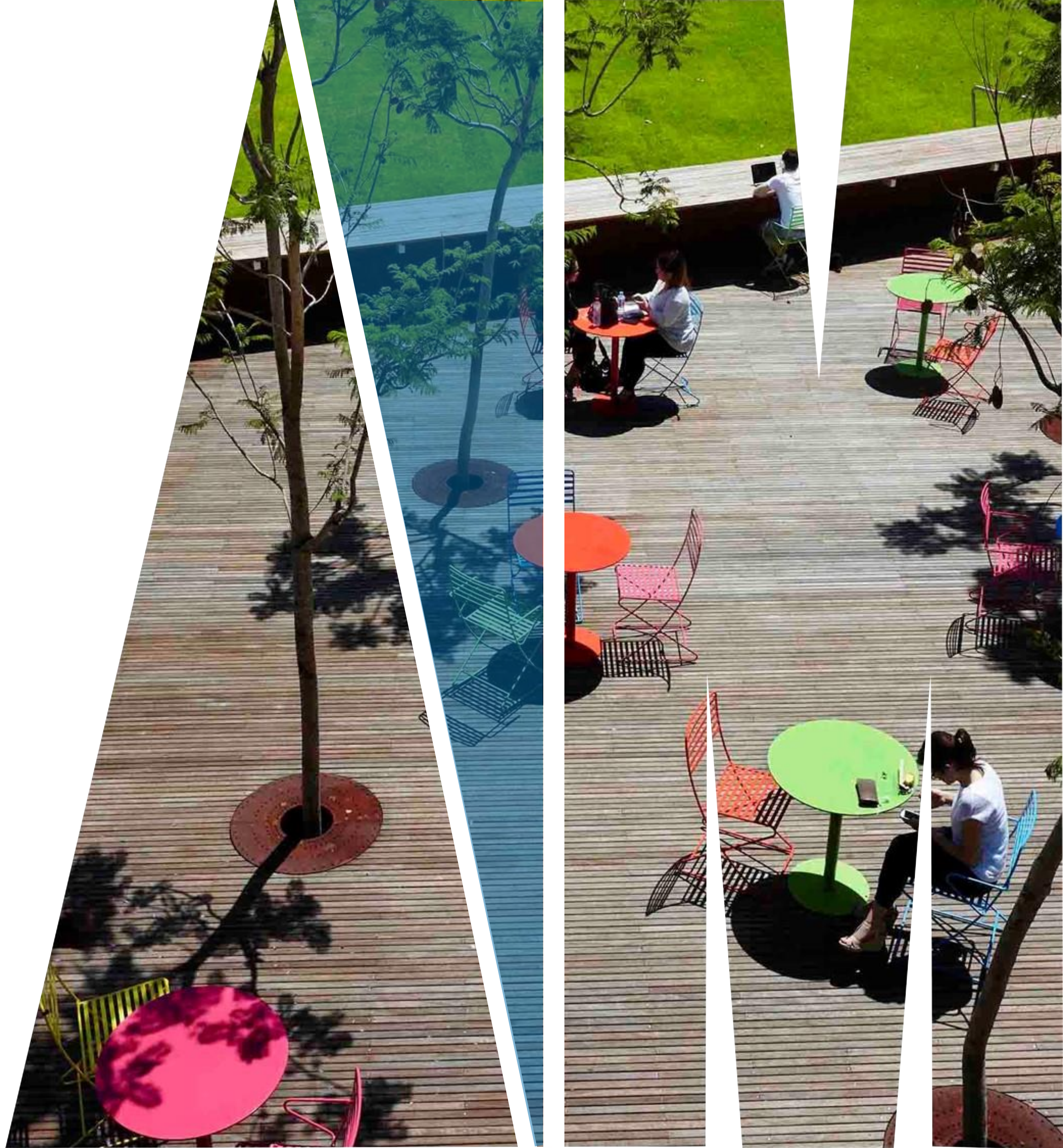
MONASH University

FIT2099 Object-Oriented Design and Implementation

Connascence and Encapsulation



MONASH University

Outline

Properties of connascence

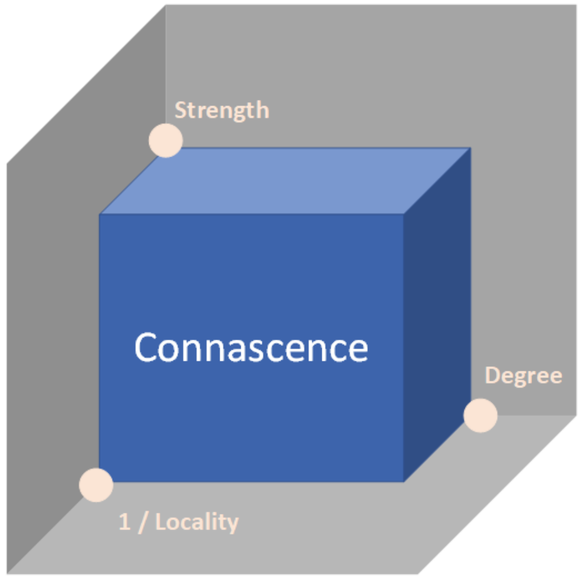
Minimizing bad connascence

Connascence and encapsulation

PROPERTIES OF

CONNASCENCE

1- Strength. The higher level of connascence, the higher the strength.



LEVELS OF CONNASCENCE

Static
- Name
- Type
- Meaning
- Position
- Algorithm

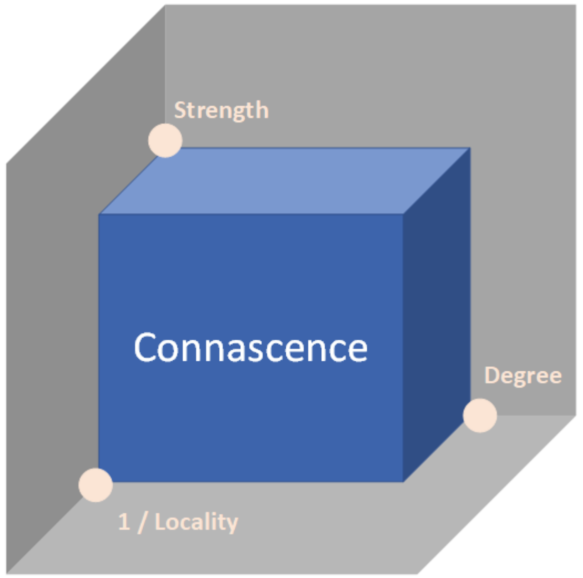
Dynamic
- Execution
- Timing
- Value
- Identity

PROPERTIES OF CONNASCENCE

1- Strength. The higher level of connascence, the higher the strength.

2- Locality. It describes how close the coupled components are.

The higher connascence locality, the better. Coupled methods that are ***in different classes*** *are much* ***worse*** *than coupled methods within* ***same class****.*



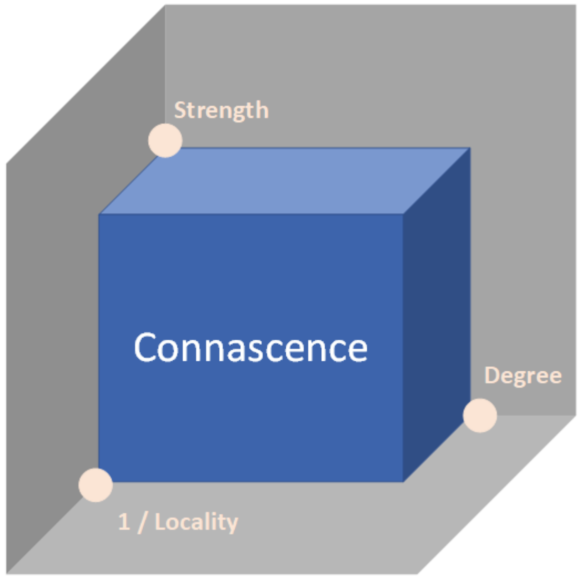
MONASH University

PROPERTIES OF CONNASCENCE

1- Strength. The higher level of connascence, the higher the strength.

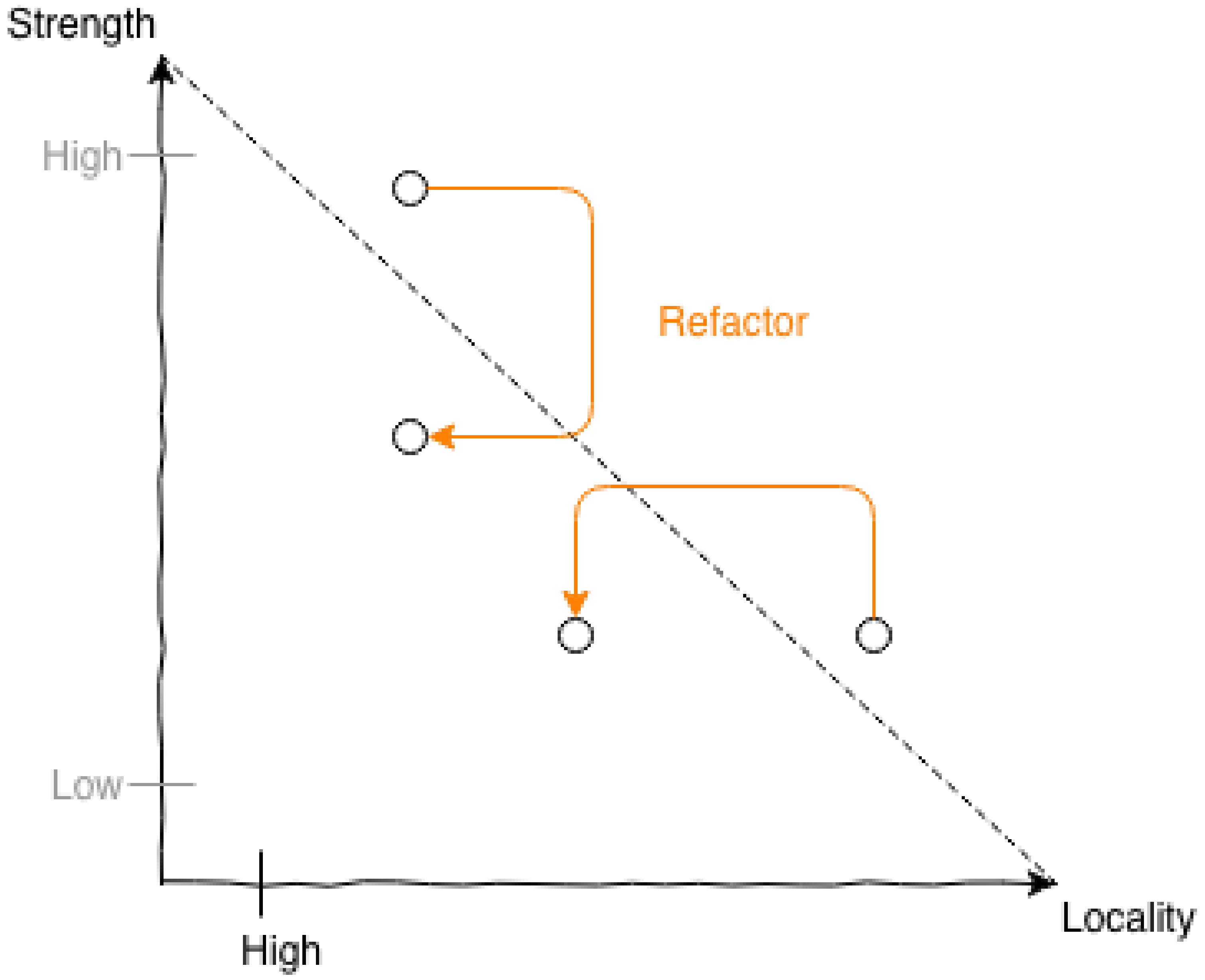
2- Locality. It describes how close the coupled components are.

*The higher connascence locality, the better. Coupled methods that are **in different classes** are much **worse** than coupled methods within **same class**.*



MONASH University

CONNASCENCE AND REFACTORING



PRACTICAL RECOMMENDATIONS ABOUT CONNASCENCE

Not all instances of connascence are equal!

In general, later-listed ones are **worse** than others.

Locality matters!

- **Within a method** -> almost (but not totally) irrelevant.
- **Between two methods in a class** -> often no big deal.
- **Two classes** -> warning warning
- **Two classes in different packages** -> WARNING WARNING!!!
- **Across application boundaries** -> keep to absolute minimum

Explicitness matters

WHAT CAN WE DO ABOUT CONNASCENCE?

1. Minimise overall amount of connascence by breaking system into ***encapsulated*** elements.
2. Minimise remaining connascence that crosses ***encapsulation boundaries*** (guideline 3 will help with this)
3. **Maximise** connascence *within encapsulation boundaries*
   – we’ll talk about “encapsulation” regarding connascence soon!

ENCAPSULATION IN OUR SMART HOME APPLICATION

The final abstract SmartDevice had the following **public** methods:

- One constructor
- Getters and setters
- display()
- turnOn()
- turnoff()

External code have no access to *anything* else.

- So those five methods are the *only* opportunities for **connascence**

Further functionality was added via **abstraction**.

Pretty well encapsulated!

ENCAPSULATION IN THE JAVA COLLECTIONS FRAMEWORK

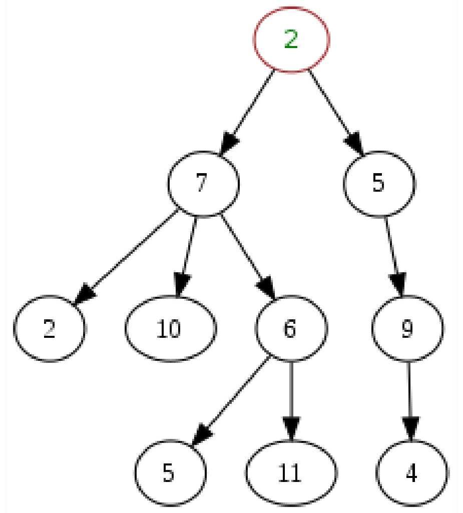
```
class TreeNode<K extends Comparable<K>, V> {

private TreeNode leftChild;
private TreeNode rightChild;

private K key;
private V data;

...

}
```



https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/tree/TreeNode.html

REDUCING CONNASCENE USING ENCAPSULATION

Using Java is not enough!

- no language is a silver bullet

Design your system carefully

Take advantage of features provided by the **language**

- access control modifiers (`private`, `protected`, etc)
- classes, packages

SIMPLE STRATEGIES TO REDUCE CONNASCENCE

Avoid public attributes

Only make methods public where necessary

  consider a policy of making everything private when you first create it

Keep the class package-private if not needed!

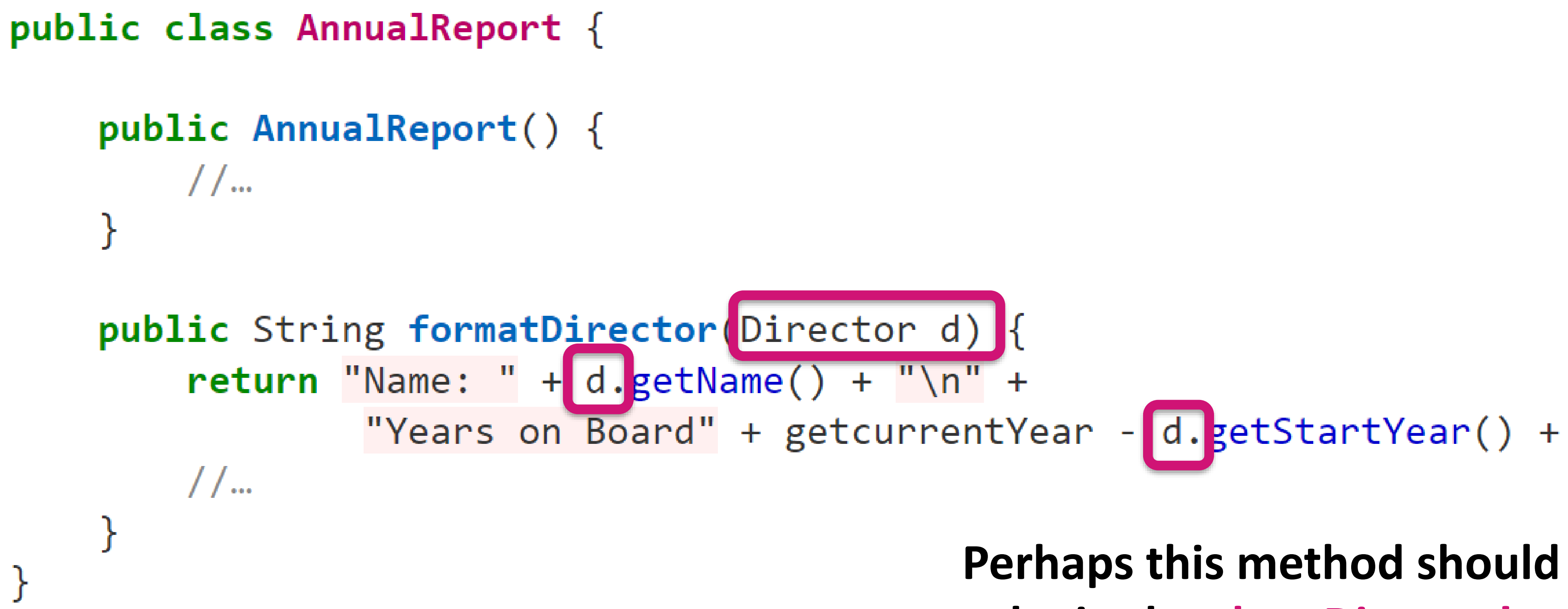
Use `protected` sparingly

  consider using methods rather than attributes

  remember that protected things are accessible to subclasses in ***other packages.***

Simple.

PUT METHODS IN THE RIGHT PLACE



```
public class AnnualReport {

    public AnnualReport() {
        //…
    }

    public String formatDirector(Director d) {
        return "Name:  " + d.getName() + "\n" +
               "Years on Board" + getcurrentYear - d.getStartYear() +
        //…
    }
}
```

Perhaps this method should be in the class Director!

AVOID SIMPLE

CONNASCENCE OF EXECUTION

```java
public class Couple {

    private Person member1 = null;
    private Person member2 = null;
    public Couple() {
    }

    public void setPerson1(Person p1) {
        member1 = p1;
    }

    public void setPerson2(Person p2) {
        member2 = p2 ;
    }

    public String toString() {
        return member1.description() + " " + member2.description();
    }
}
```

AVOID SIMPLE
CONNASCENCE OF EXECUTION

```
public class Couple {

    private Person member1 = null;
    private Person member2 = null;
    public Couple() {
    }

    public void setPerson1(Person p1) {
        member1 = p1;
    }

    public void setPerson2(Person p2) {
        member2 = p2 ;
    }

    public String toString() {
        return member1.description() + " " + member2.description();
    }
}
```

AVOID SIMPLE

CONNASCENCE OF EXECUTION

```
public class Couple {

    private Person member1 = null;
    private Person member2 = null;
    public Couple() {
    }

    public void setPerson1(Person p1) {
        member1 = p1;
    }

    public void setPerson2(Person p2) {
        member2 = p2 ;
    }

    public String toString() {
        return member1.description() + " " + member2.description();
    }
}
```

MINIMISE

APIs

Application programming interface: that part of a class/package/module that is accessible from outside

aka *public interface*

The smaller and less complicated the interface is, the fewer opportunities for connascence there are

Make things private if you can

Summary

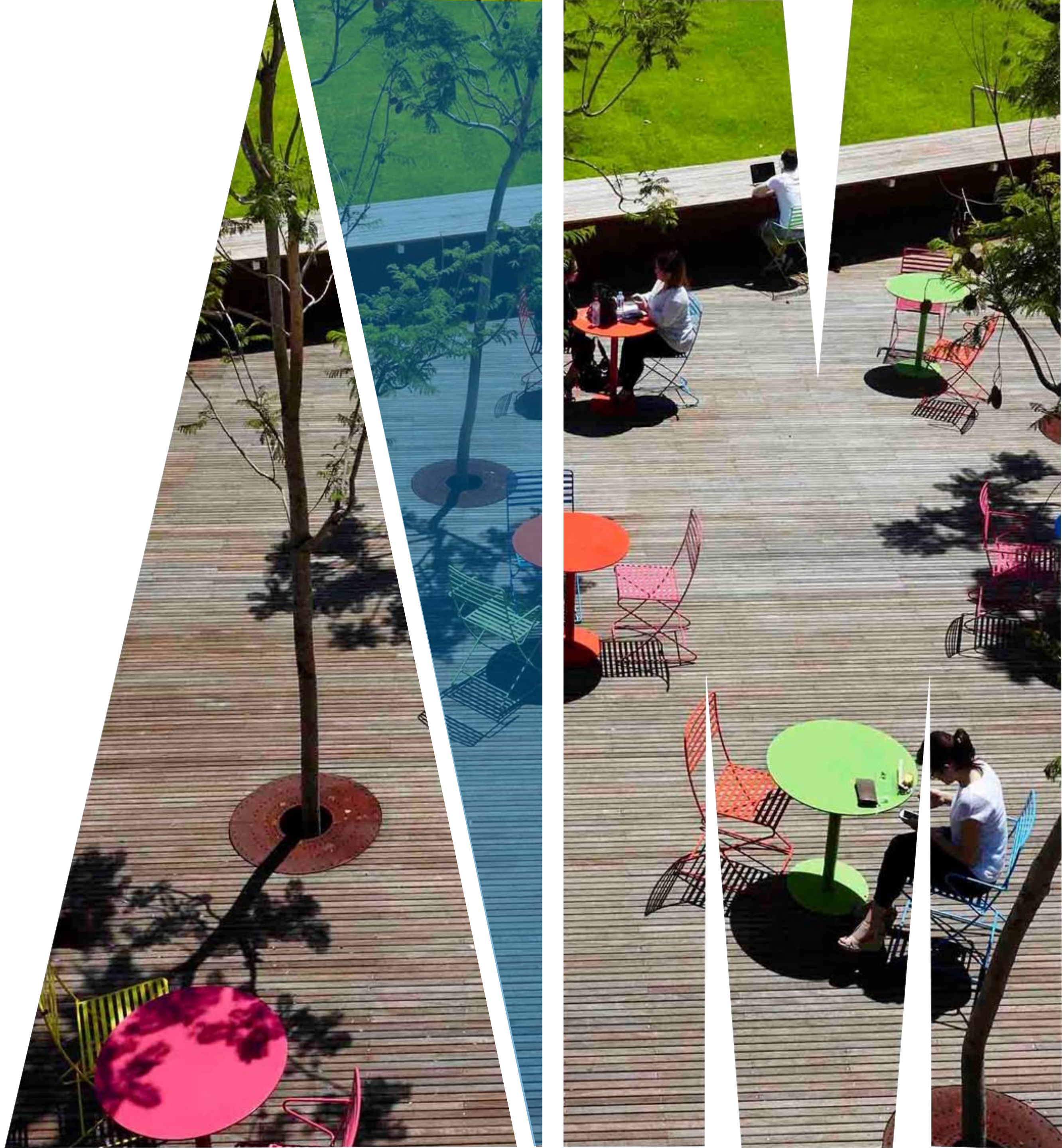
Properties of connascence

Minimizing bad connascence

Connascence and encapsulation

MONASH University

Thanks



MONASH University