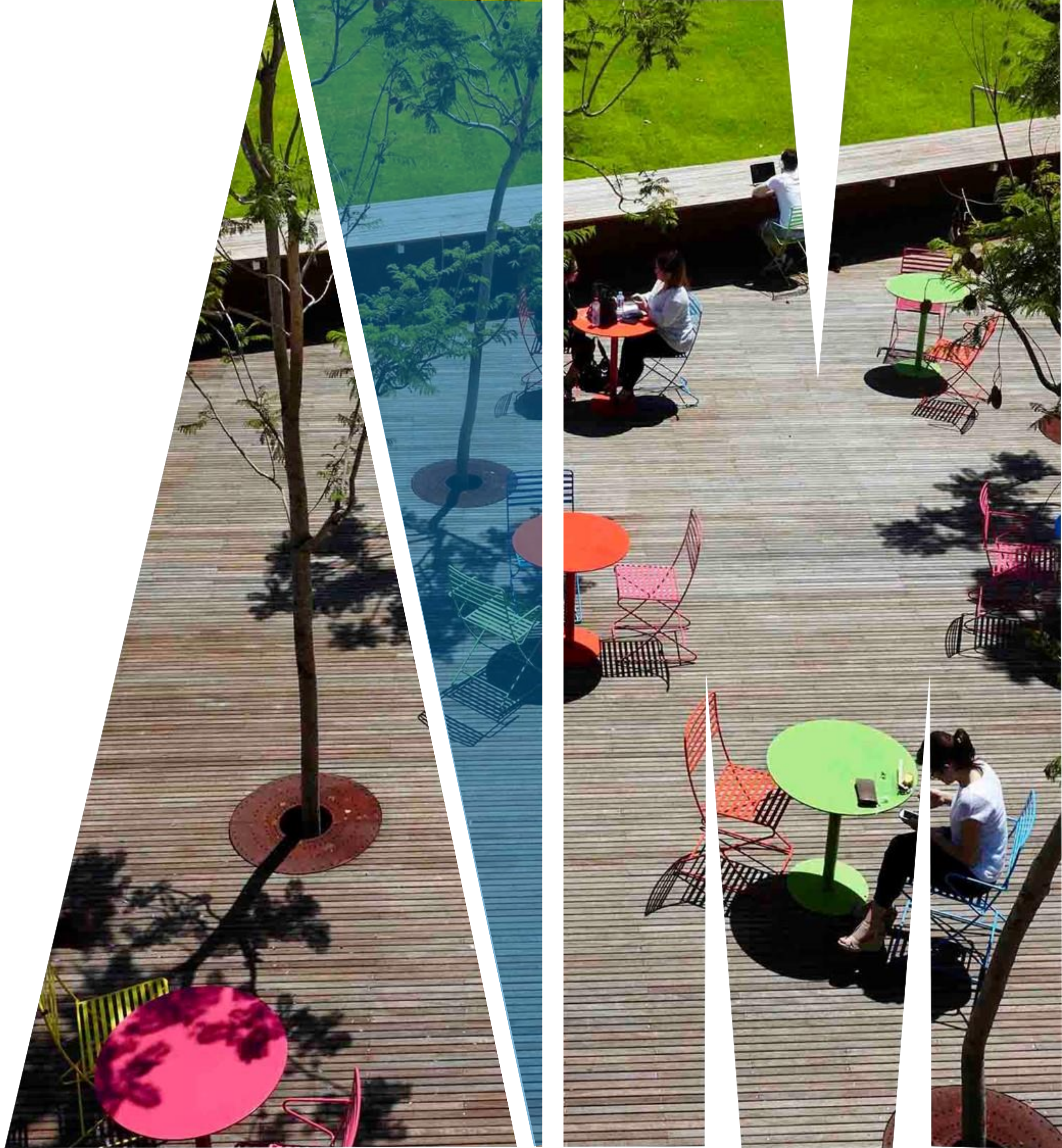
FIT2099 Object-Oriented Design and Implementation

Encapsulation in Java (packages and modules)



Outline

Encapsulation boundaries

Encapsulation in Java

- Packages and modules

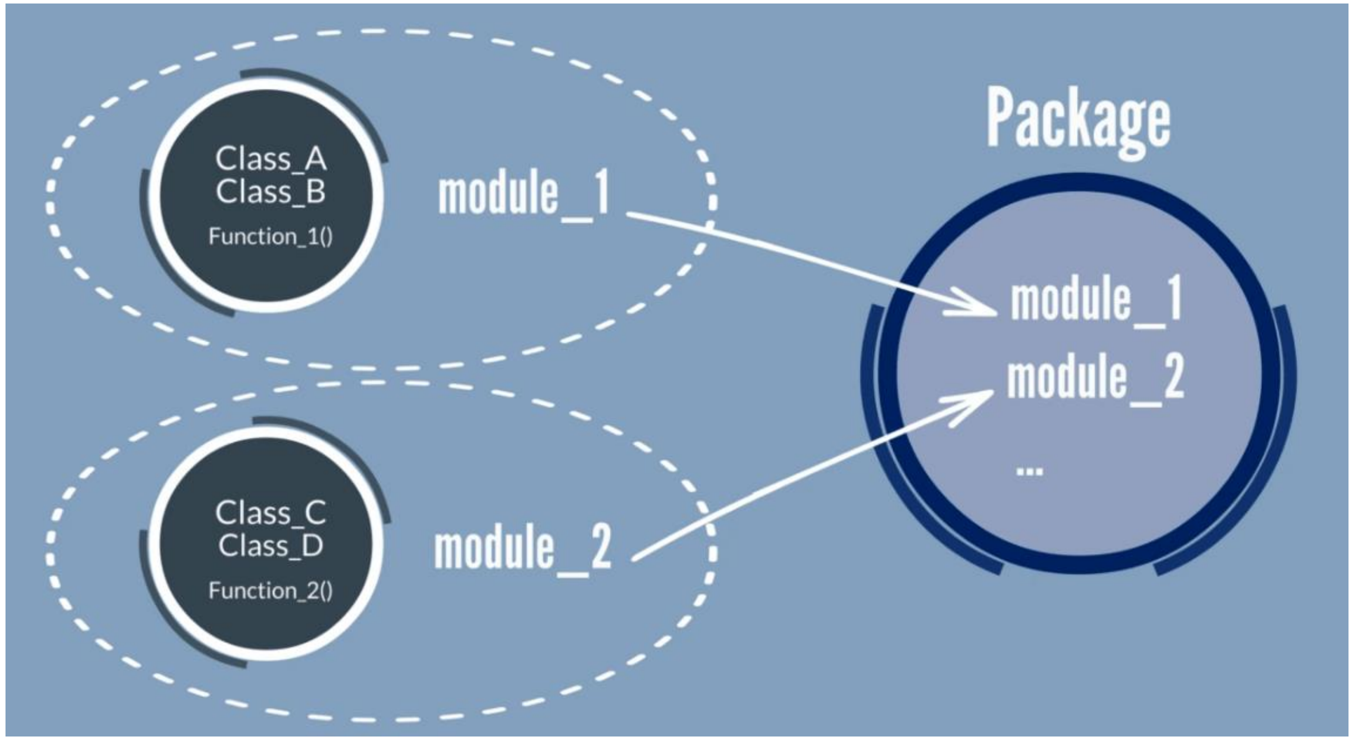
MONASH University

ENCAPSULATION
BOUNDARIES

An **encapsulation boundary** is simply something across which visibility can be restricted

- the class
- the package
- the module
- even the scope defined

within methods by curly braces {}



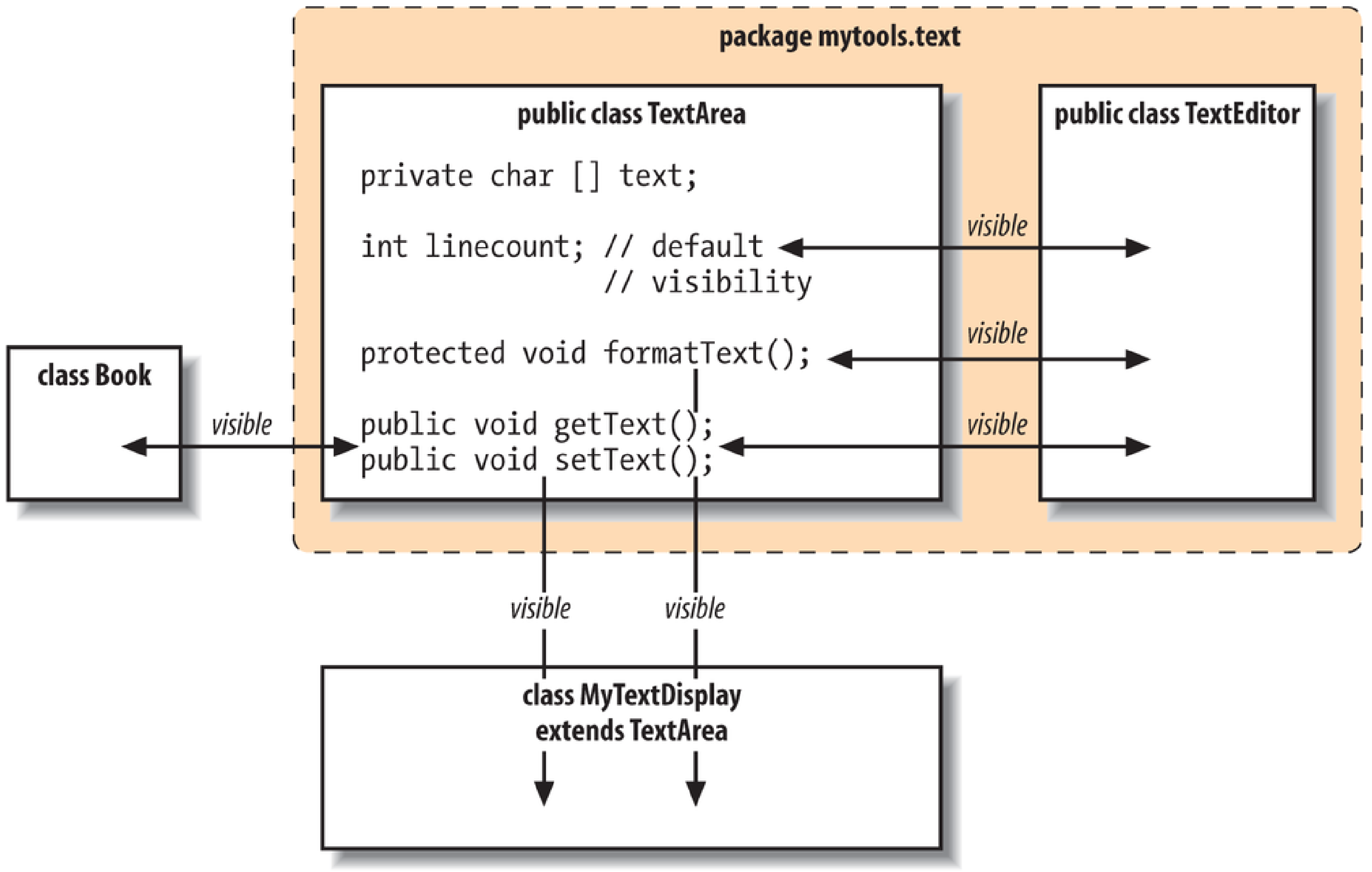
MONASH University

ENCAPSULATION
BOUNDARIES

Any method call or attribute accesses that that is not in the same class (or package) **crosses** an encapsulation boundary

You want to **minimize** these accesses - that's what we mean by "ReD"

So… expose (i.e. make public) the methods/attributes that client code really needs, and hide everything else



ReD: reducing dependency

ENCAPSULATION IN JAVA

Java was ***designed*** to encapsulate

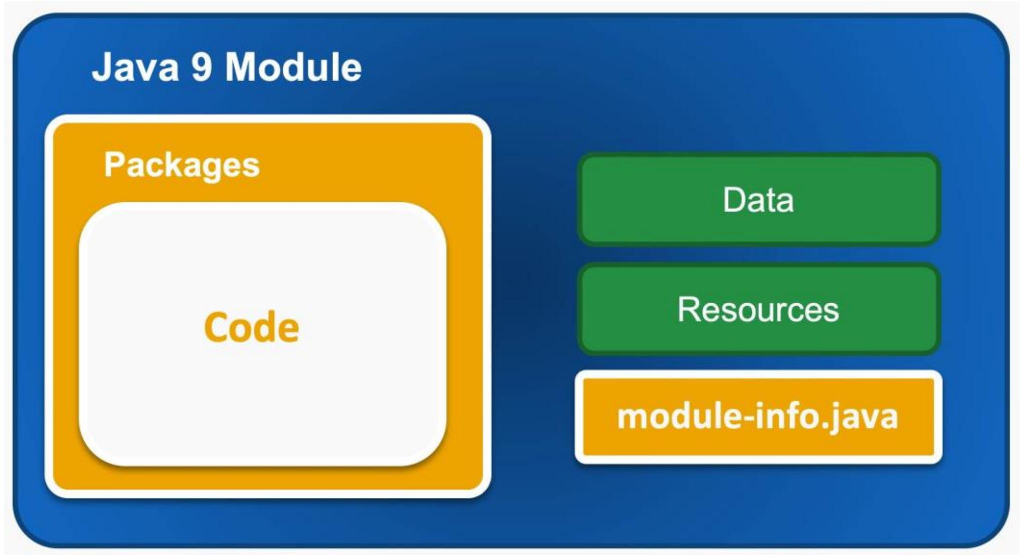
- as are many other OO languages
- and non-OO ones too!

Basic unit of Java programs is **the class**

- can group classes into ***packages***
- can group packages into ***modules*** (as of Java 9)

Can restrict access to *anything* in the class to:

- within the class only (**private**)
- within the package only (**no access modifier** - default)
- only to subclasses and within the package (**protected**)
- no restrictions (**public**)



THE MODULES

Introduced in Java 9

Essentially, **a collection of Java packages** organized in the usual way

Includes a ***module descriptor*** file (module-info.java) that specifies

- the **name** of the module
- any other modules that this module **depends on**
- which packages are **public**
- any **services** this module offers
- which services this module **consumes**
- which other classes may apply **reflection** to packages in this module

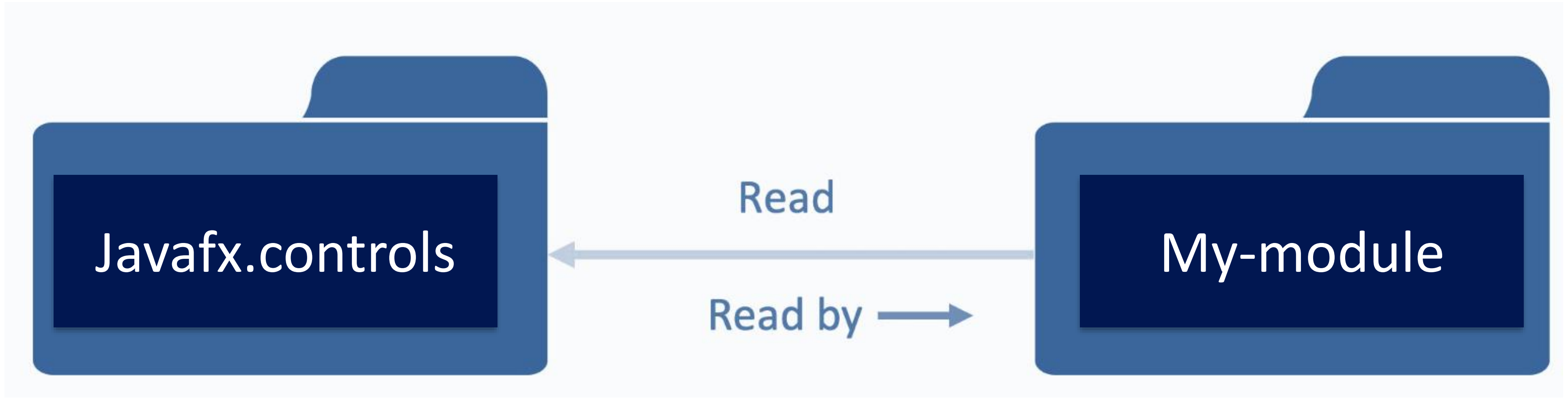
THE module-info.java FILE

```
module my-module {
   requires javafx.controls;
   exports my.program.package;
   exports my.program.package
      to other-module;
   provides someInterface
      with my.program.Implementation;
}
```

MONASH University

module-info.java

```
module my-module {
    requires javafx.controls;
    exports my.program.package;
}
```



DEPENDENCY

module-info.java

```
module my-module {
   requires javafx.controls;
   exports my.program.package;
   exports my.program.package
      to other-module;
   provides someInterface
      with my.program.Implementation;
}
```

Make specific packages visible

module-info.java

```
module my-module {
    requires javafx.controls;
    exports my.program.package;
    exports my.program.package
        to other-module;
    provides someInterface
        with my.program.Implementation;
}
```

Make specific packages visible (to specific module)

MONASH University

module-info.java

```
module my-module {
   requires javafx.controls;
   exports my.program.package;
   exports my.program.package
      to other-module;
   provides someInterface
      with my.program.Implementation;
}
```

Implements this interface

MONASH University

POTENTIAL DOWNSIDES OF CAREFUL ENCAPSULATION

More work initially
Requires **careful thought**

Payoff later
- and it is a **BIG** payoff
- the **larger the codebase**,
  … **the bigger the benefit**

MONASH University

Summary

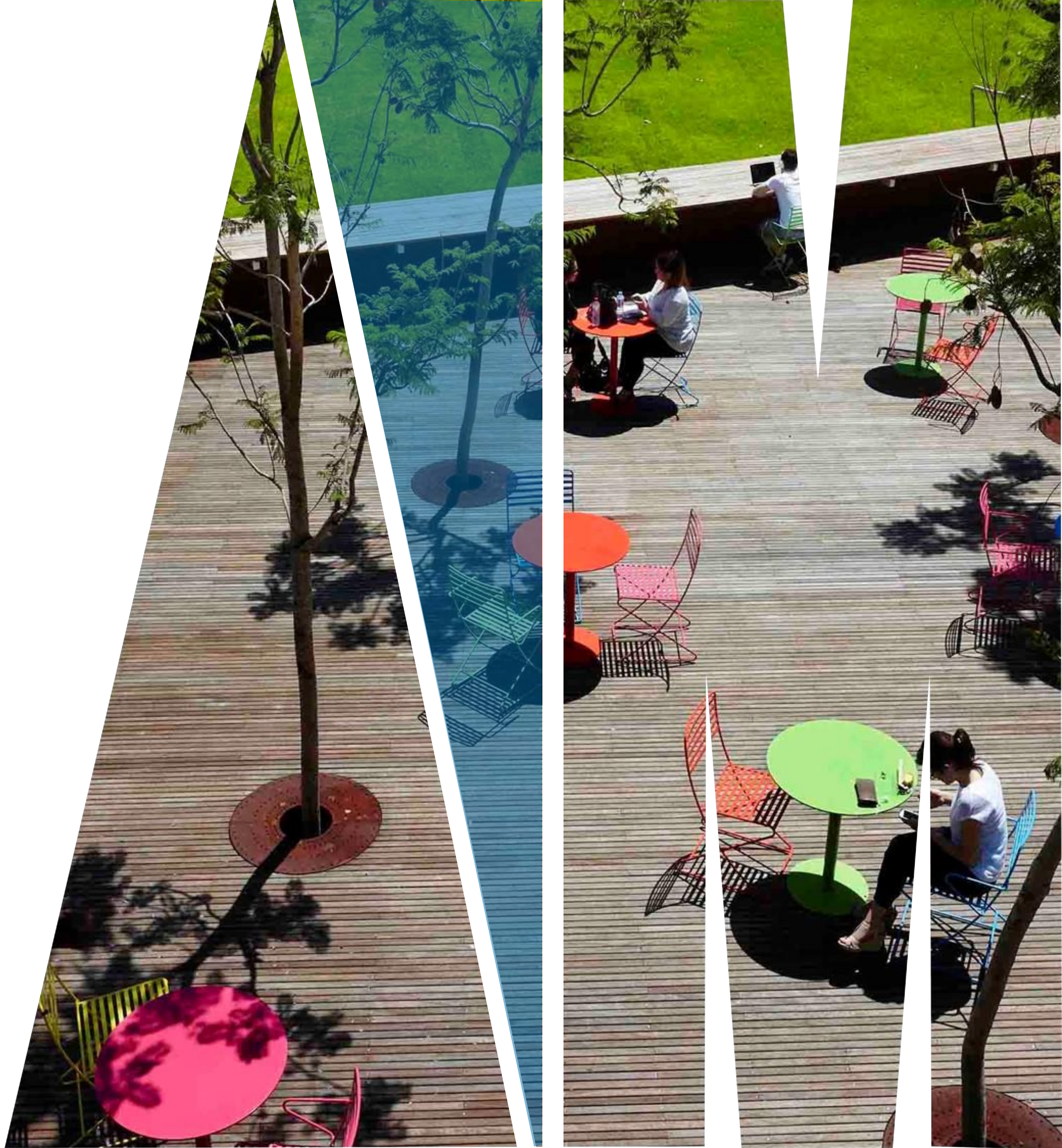
Encapsulation boundaries

Encapsulation in Java

- Packages and modules

MONASH University

Thanks



MONASH University