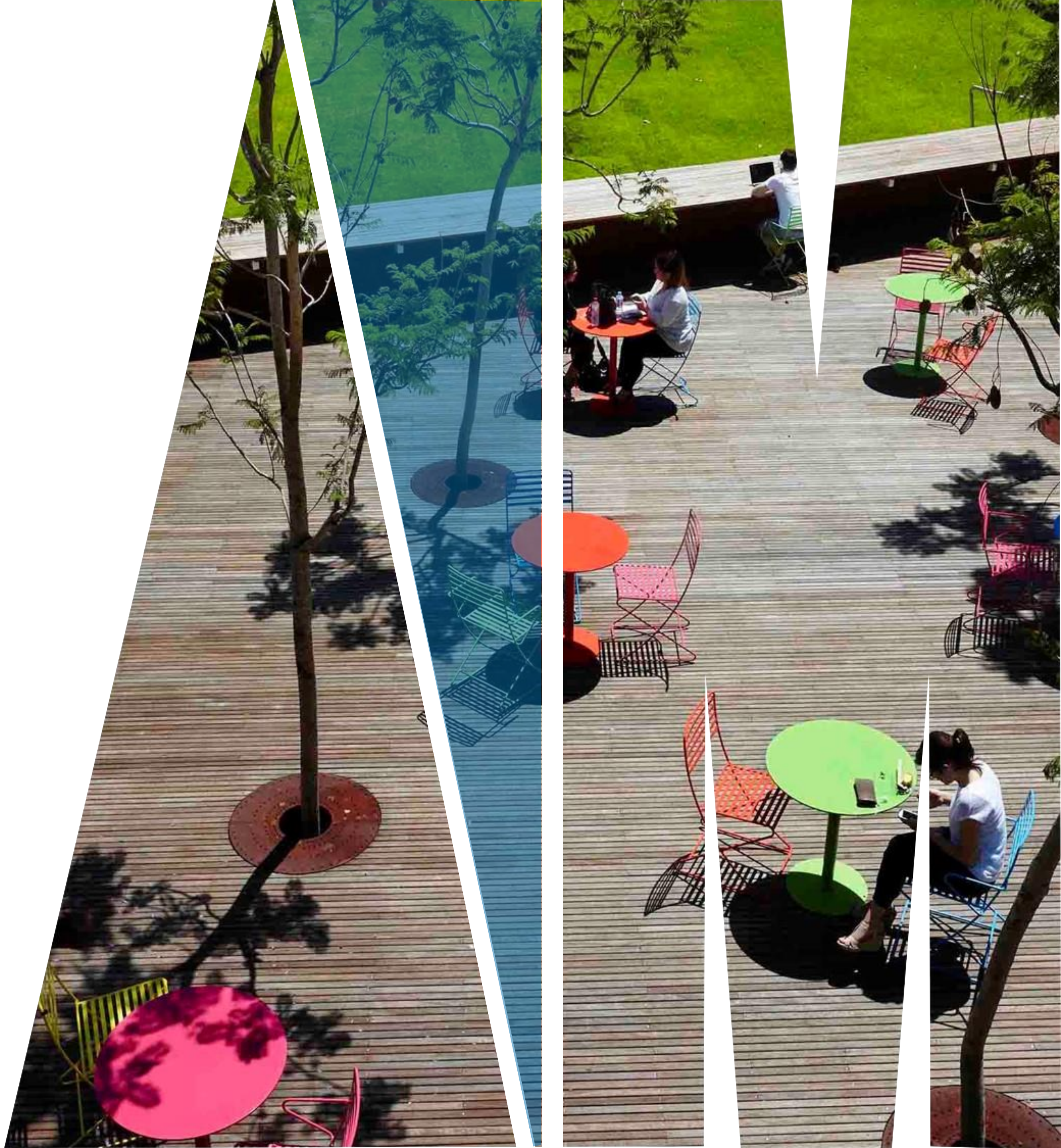
MONASH University

FIT2099 Object-Oriented Design and Implementation

Inheritance



Outline

Inheritance

UML representation

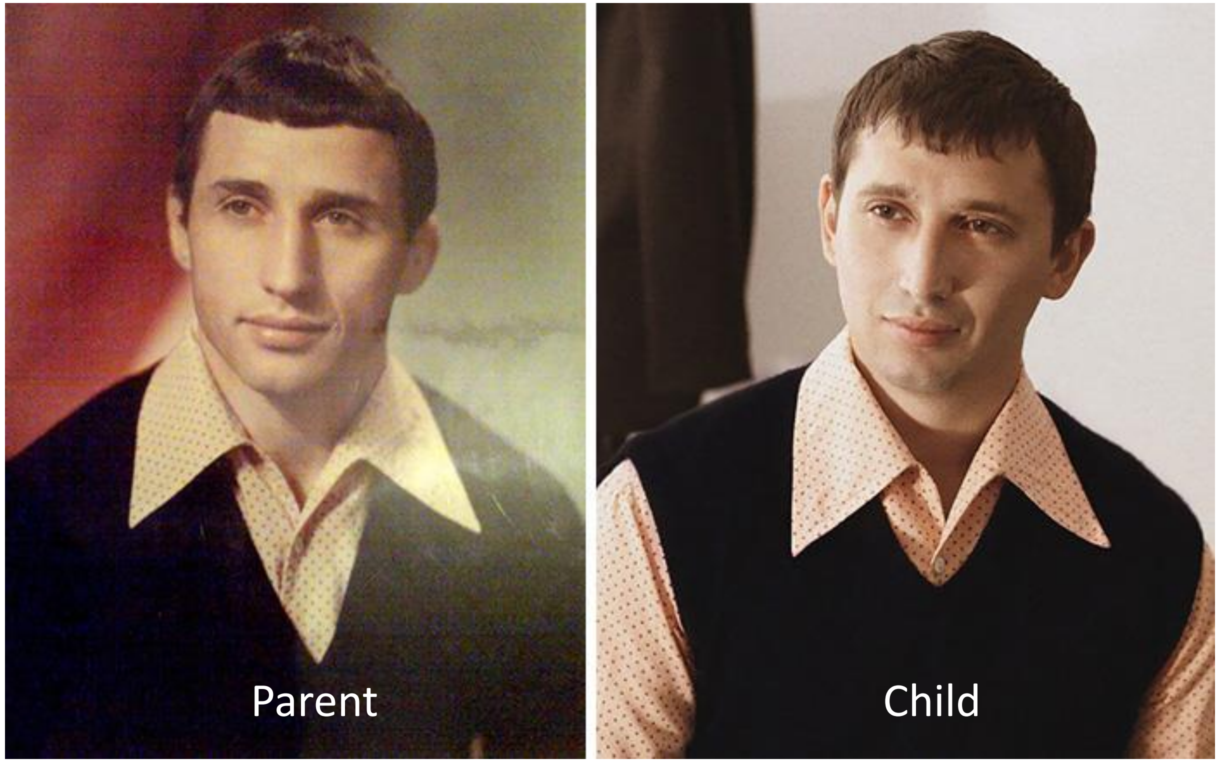
Java Syntax

Access modifiers (protected)

WHAT IS
INHERITANCE?

Inheritance is a mechanism in which **one class acquires the 'properties' of another class**.

With inheritance, we can reuse the fields and methods of the existing class. Hence, it facilitates **reusability** and is an important concept of OO design.



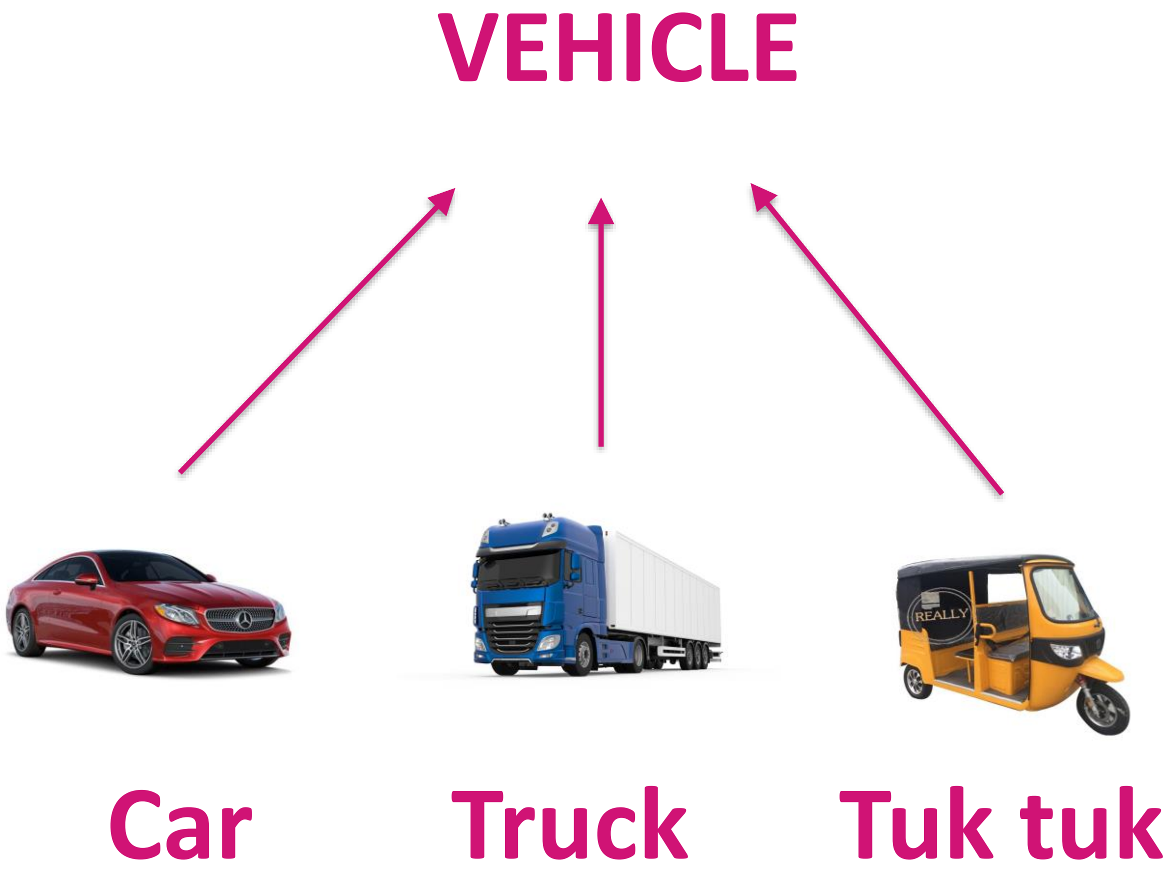
For example, children inherit the traits of their parents.

WHAT IS
INHERITANCE?

Inheritance requires at least two classes.

subclass (child) - the class that inherits from another class
superclass (parent) - the class being inherited from

In the example below, the **Car class (subclass)** inherits the attributes and methods from the **Vehicle class (superclass)**


VEHICLE

Car

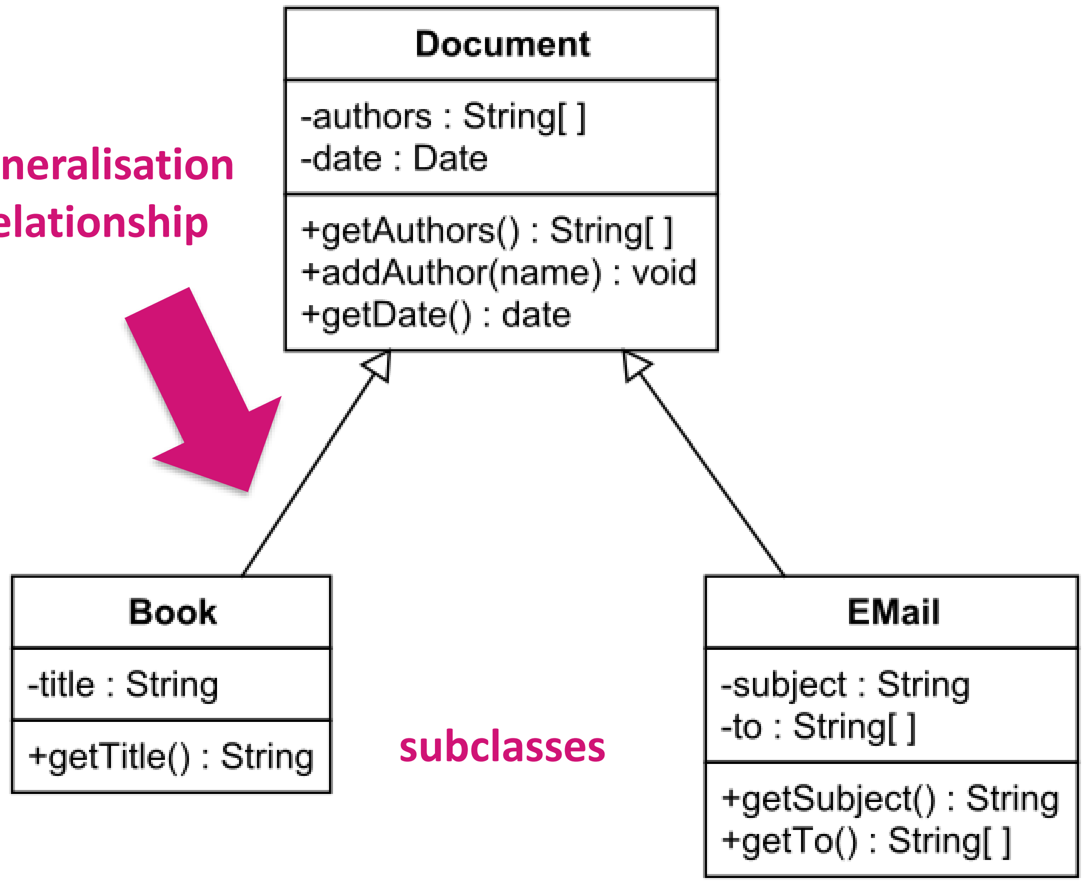
Truck

Tuk tuk

WHAT IS INHERITANCE IN UML?

The inheritance relationships in UML match up very closely with inheritance in Java.

Generalisation: A class *extends* another class.



superclass

Generalisation relationship

subclasses

Document

-authors : String[ ]
-date : Date

+getAuthors() : String[ ]
+addAuthor(name) : void
+getDate() : date

Book

-title : String

+getTitle() : String

EMail

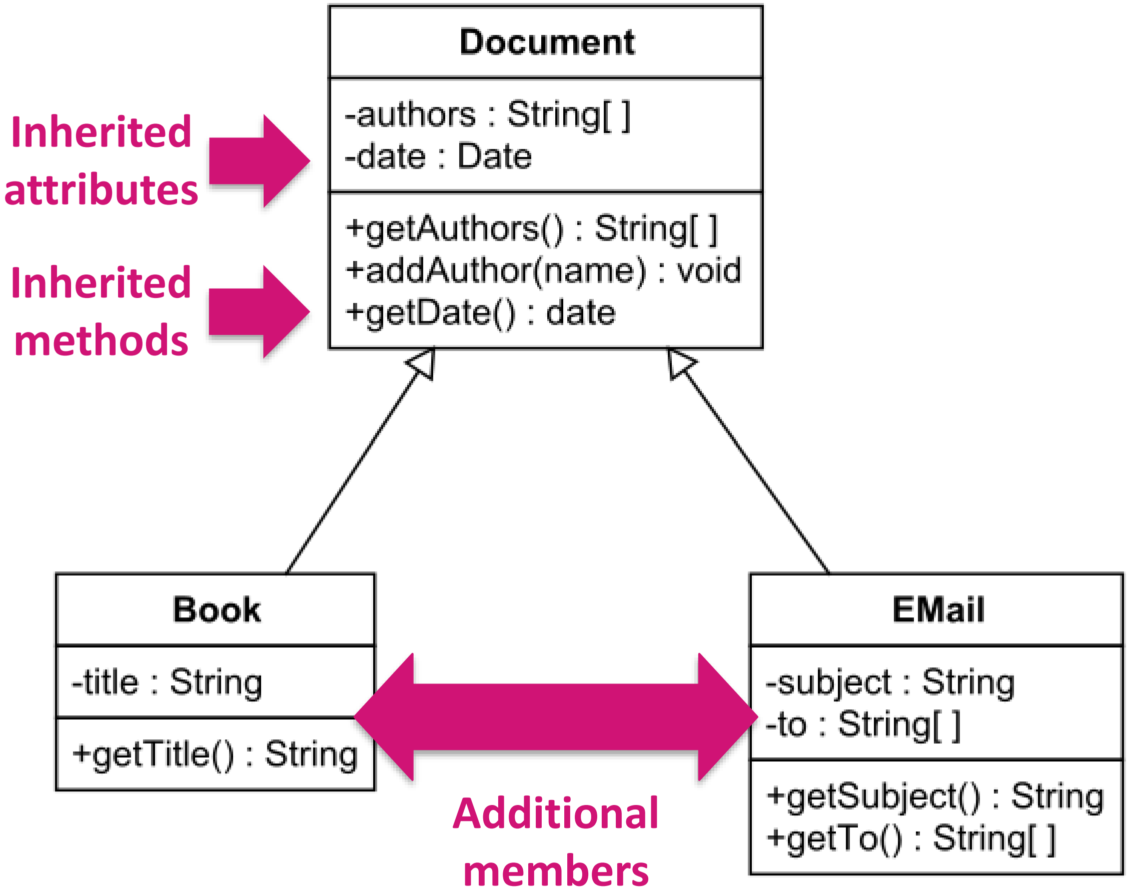
-subject : String
-to : String[ ]

+getSubject() : String
+getTo() : String[ ]

WHAT IS
INHERITANCE IN UML?

For example, the **Book** class might extend the **Document** class, which also might include the **Email** class.

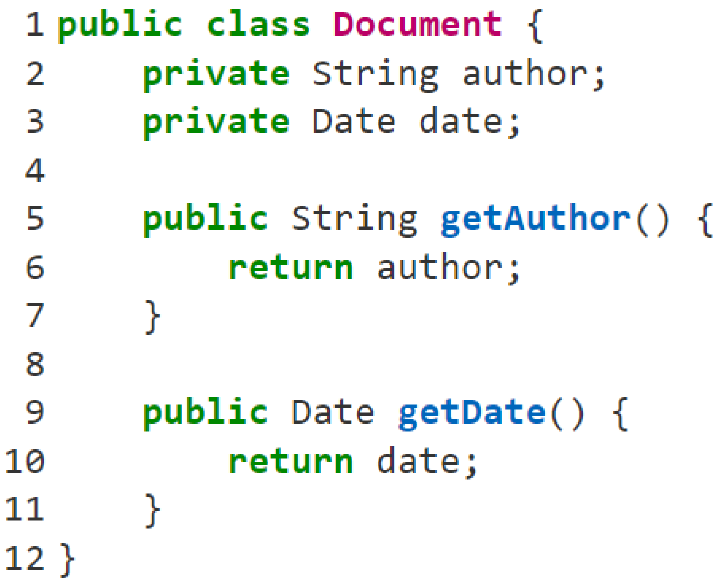
The Book and Email classes inherit the attributes and methods of the Document class (possibly modifying the methods), but might add additional additional and methods.



WHAT IS
INHERITANCE IN JAVA?

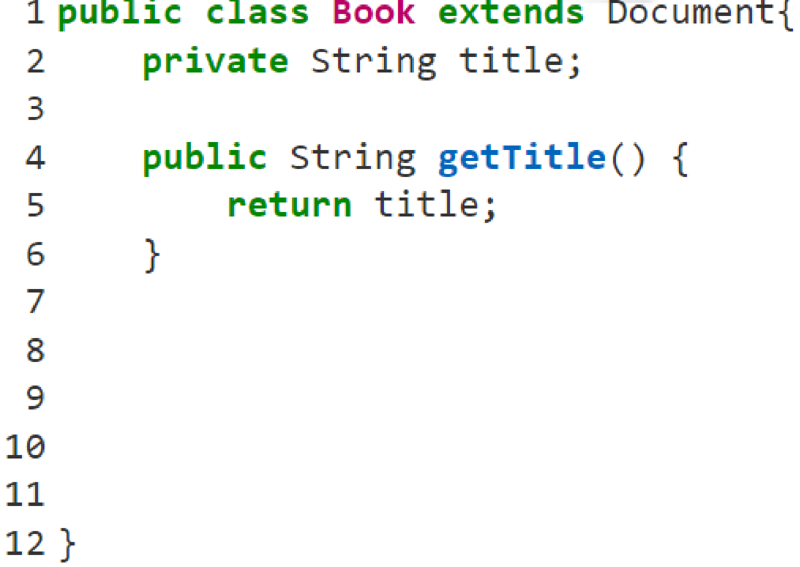
In Java, to inherit from a class, use the **extends** keyword. Simplified example:

superclass



```java
public class Document {
    private String author;
    private Date date;

    public String getAuthor() {
        return author;
    }

    public Date getDate() {
        return date;
    }
}
```

subclass

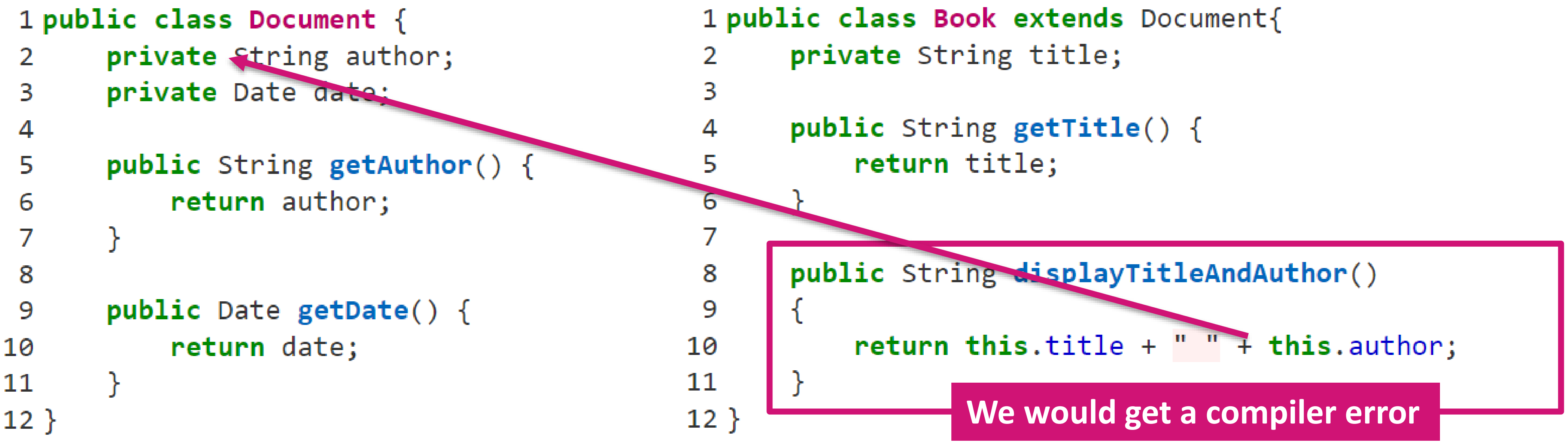


```java
public class Book extends Document{
    private String title;

    public String getTitle() {
        return title;
    }





}
```

HOW TO ACCESS THE MEMBERS OF THE SUPERCLASS?

It is **NOT** possible to access members of the superclass which have '**private**' access from the subclass.

superclass **subclass**



```java
public class Document {
    private String author;
    private Date date;

    public String getAuthor() {
        return author;
    }

    public Date getDate() {
        return date;
    }
}
```

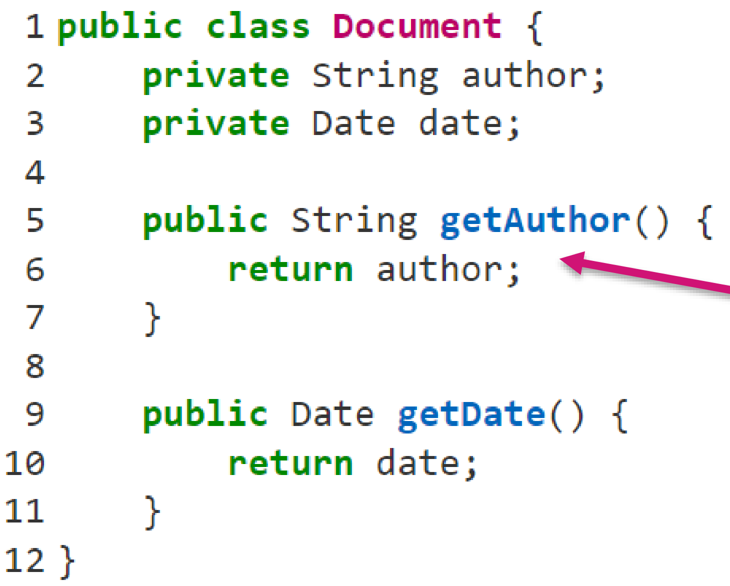
```java
public class Book extends Document{
    private String title;

    public String getTitle() {
        return title;
    }

    public String displayTitleAndAuthor()
    {
        return this.title + " " + this.author;
    }
}
```

We would get a compiler error

HOW TO ACCESS THE MEMBERS OF THE SUPERCLASS?

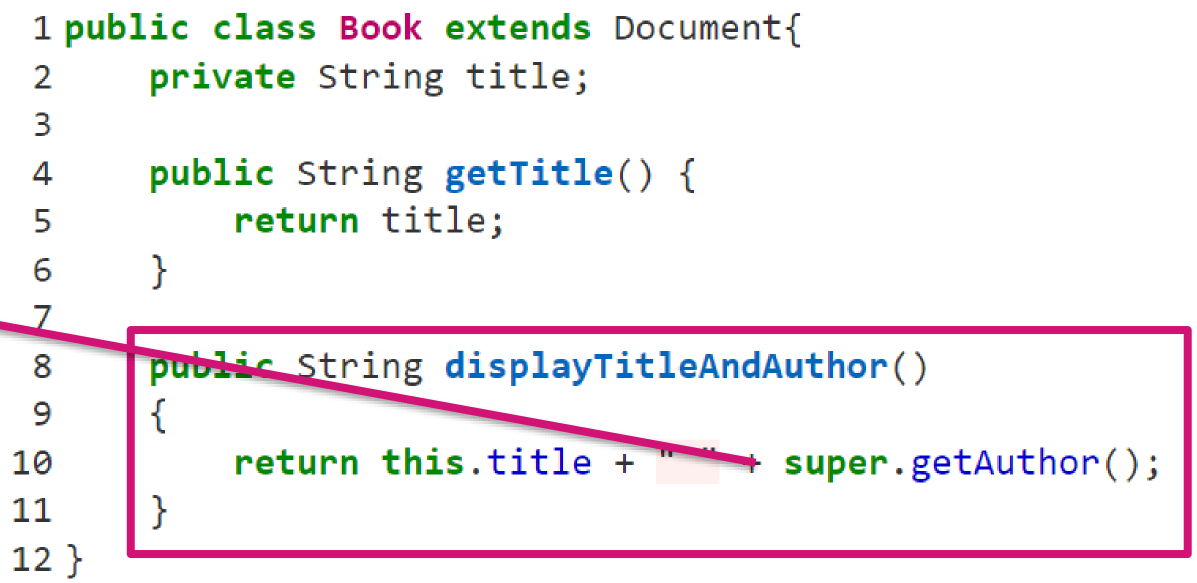
The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, **an instance of parent class is created implicitly** which is referred by super reference variable.

superclass



```java
public class Document {
    private String author;
    private Date date;

    public String getAuthor() {
        return author;
    }

    public Date getDate() {
        return date;
    }
}
```

subclass



```java
public class Book extends Document{
    private String title;

    public String getTitle() {
        return title;
    }

    public String displayTitleAndAuthor()
    {
        return this.title + " " + super.getAuthor();
    }
}
```

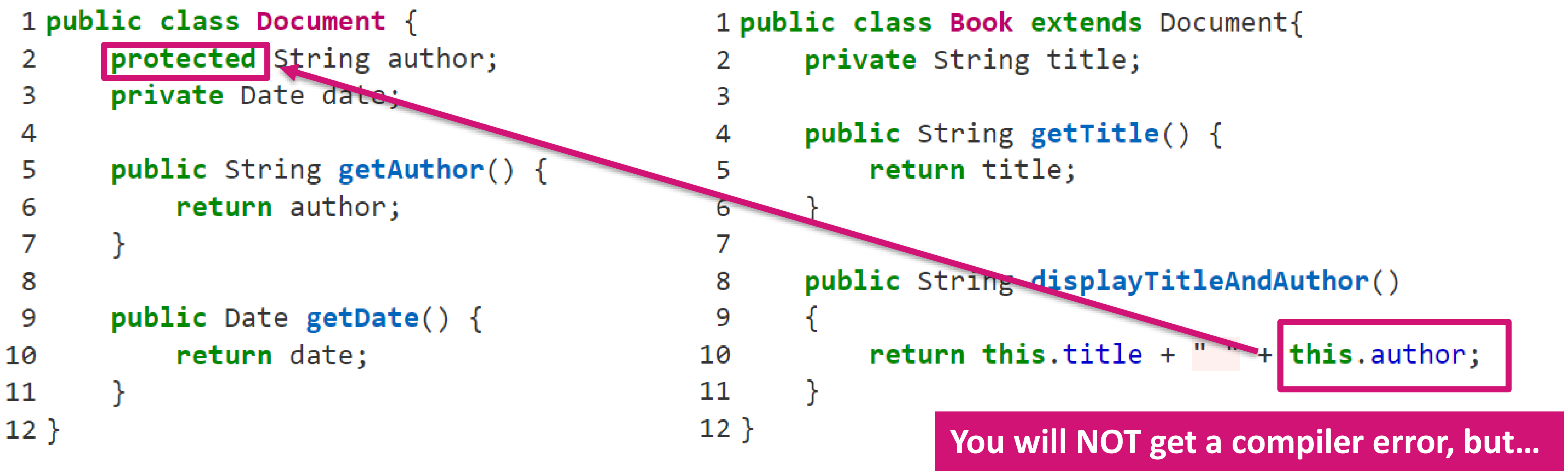
Not recommended, especially with attributes

HOW TO ACCESS MEMBERS OF THE SUPERCLASS USING THE PROTECTED KEYWORD

The **protected** keyword is an access modifier used for attributes, methods and constructors, making them accessible in the same package (pretty much package public) and **subclasses**.

superclass

subclass



```java
public class Document {
    protected String author;
    private Date date;

    public String getAuthor() {
        return author;
    }

    public Date getDate() {
        return date;
    }
}
```

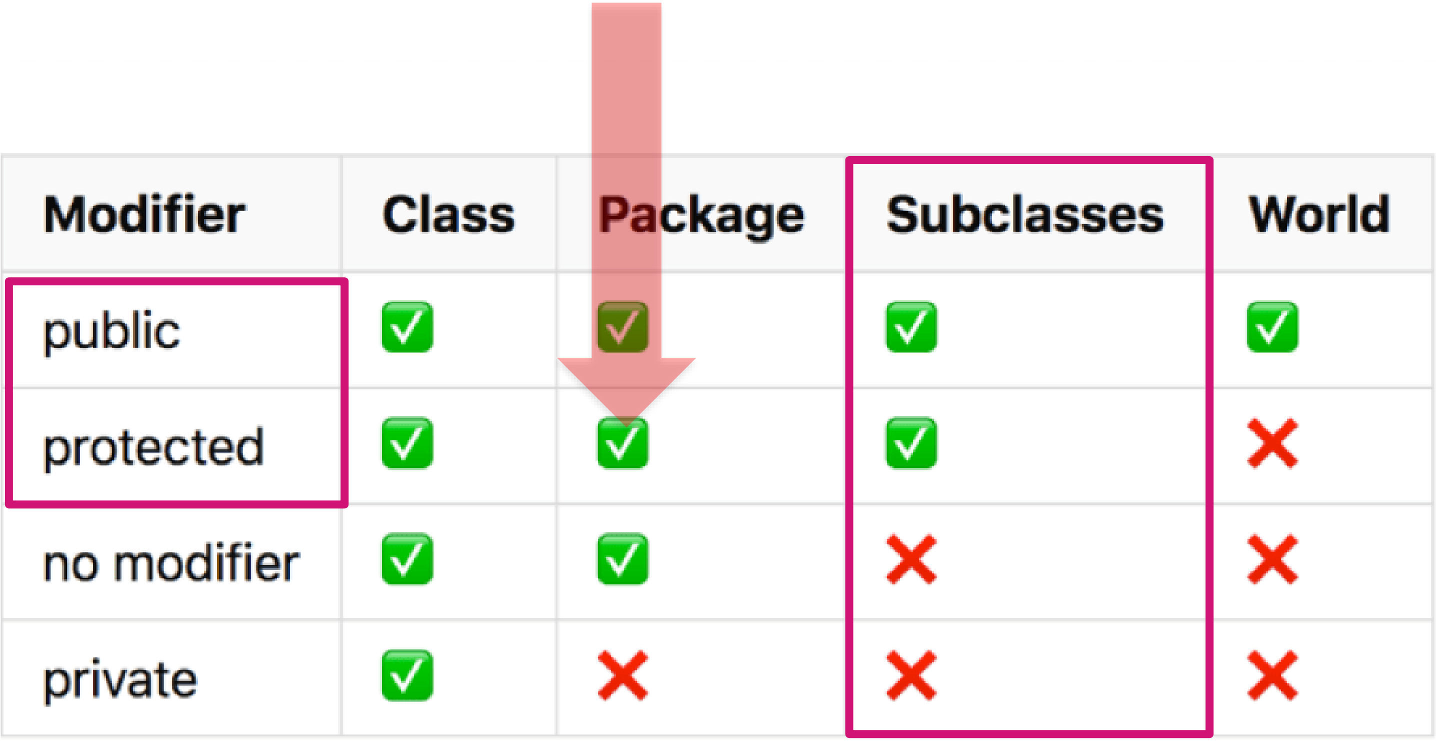
```java
public class Book extends Document{
    private String title;

    public String getTitle() {
        return title;
    }

    public String displayTitleAndAuthor()
    {
        return this.title + " " + this.author;
    }
}
```

You will NOT get a compiler error, but…

HOW TO ACCESS MEMBERS OF THE SUPERCLASS
USING THE PROTECTED KEYWORD

Not recommended, especially with attributes

The **protected** keyword is an access modifier used for attributes, methods and constructors, making them accessible in the same package (pretty much package public) and **subclasses**.

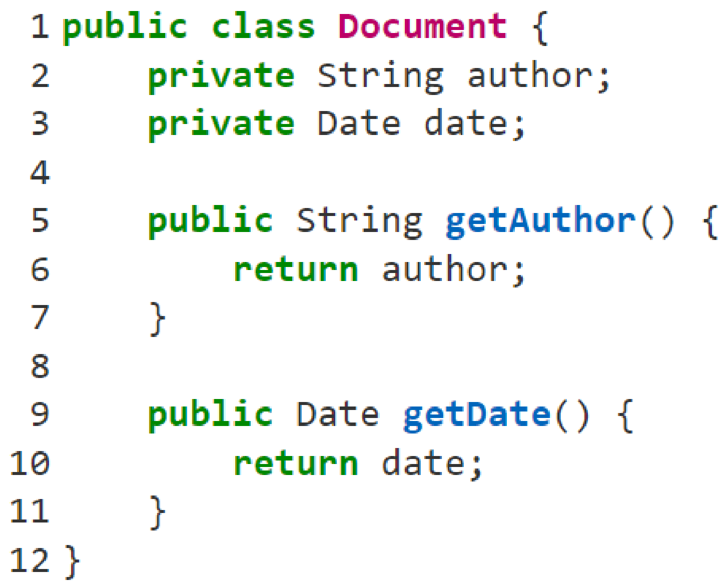


Modifier	Class	Package	Subclasses	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

WHAT ABOUT CALLING CONSTRUCTORS FROM SUBCLASSES?

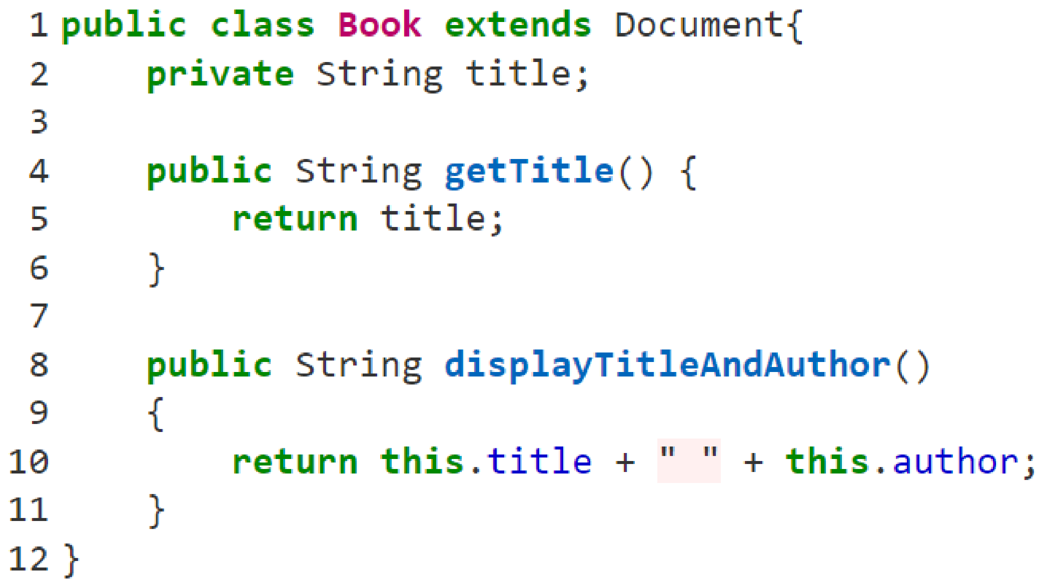
A class that extends another class **does not inherit its constructors**. However, the subclass must call a constructor in the superclass inside of its subclass constructors.

superclass



```java
public class Document {
    private String author;
    private Date date;

    public String getAuthor() {
        return author;
    }

    public Date getDate() {
        return date;
    }
}
```

subclass



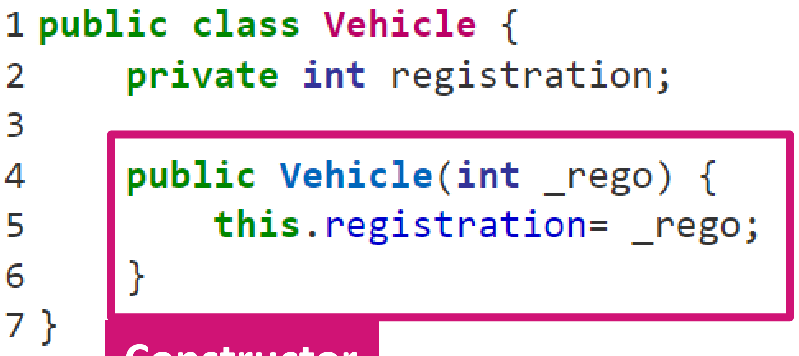
```java
public class Book extends Document{
    private String title;

    public String getTitle() {
        return title;
    }

    public String displayTitleAndAuthor()
    {
        return this.title + " " + this.author;
    }
}
```

NOTE: If a constructor does not invoke a superclass constructor, Java does so implicitly. But what if a class is declared without a constructor? In this case, Java implicitly adds a constructor to the class. This default constructor does nothing but invoke the superclass constructor.

WHAT ABOUT CALLING CONSTRUCTORS FROM SUBCLASSES?

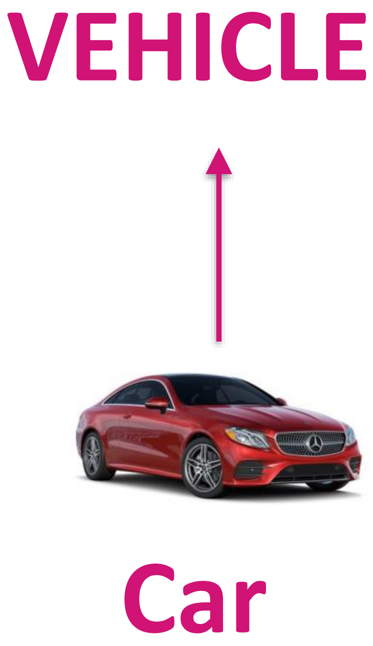
Another example:

superclass



```java
public class Vehicle {
    private int registration;

    public Vehicle(int _rego) {
        this.registration= _rego;
    }
}
```

Constructor



VEHICLE

Car

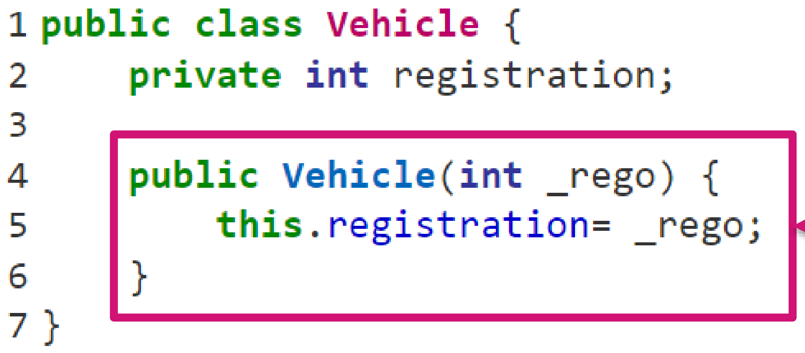
WHAT ABOUT CALLING CONSTRUCTORS FROM SUBCLASSES?

VEHICLE

Car

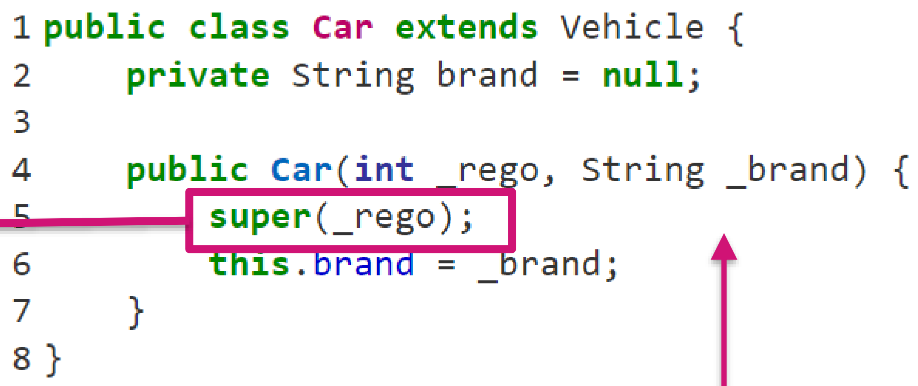
Another example:

superclass



```
public class Vehicle {
    private int registration;

    public Vehicle(int _rego) {
        this.registration= _rego;
    }
}
```

subclass

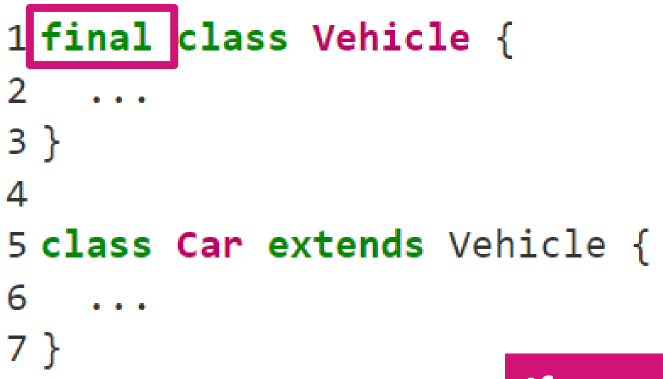


```
public class Car extends Vehicle {
    private String brand = null;

    public Car(int _rego, String _brand) {
        super(_rego);
        this.brand = _brand;
    }
}
```

You can add parameters to the constructor of the subclass

THE
FINAL KEYWORD

If you don't want other classes to inherit from a class, use the **final** keyword:



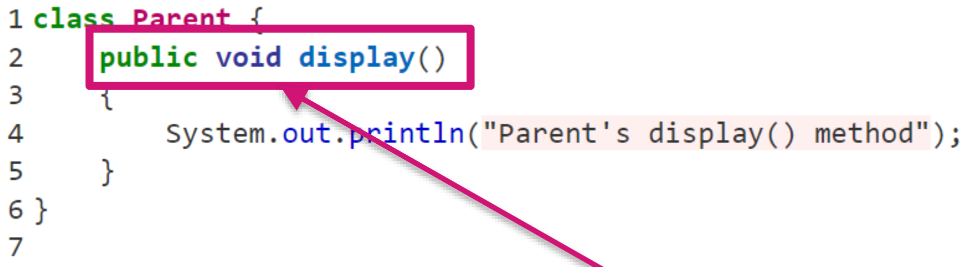
```
final class Vehicle {
  ...
}

class Car extends Vehicle {
  ...
}
```

If you try to inherit from a final class, Java will throw an error

METHOD
OVERRIDING

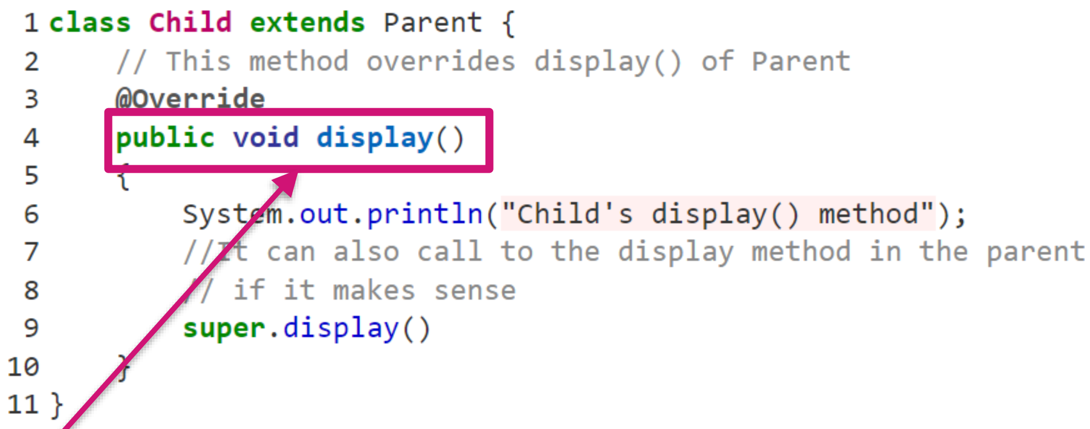
Overriding is a feature that allows a subclass or child class to provide **a specific implementation of a method** that is already provided by one of its super-classes or parent classes.

superclass



```
class Parent {
    public void display()
    {
        System.out.println("Parent's display() method");
    }
}

```

subclass

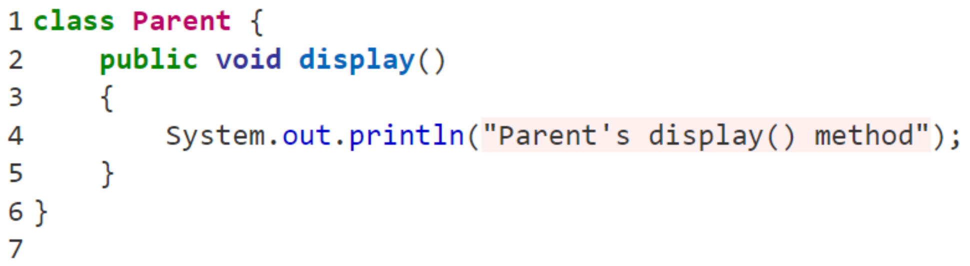


```
class Child extends Parent {
    // This method overrides display() of Parent
    @Override
    public void display()
    {
        System.out.println("Child's display() method");
        //It can also call to the display method in the parent
        // if it makes sense
        super.display()
    }
}
```

The method in the subclass has to have the same signature (same name, parameters and return type)

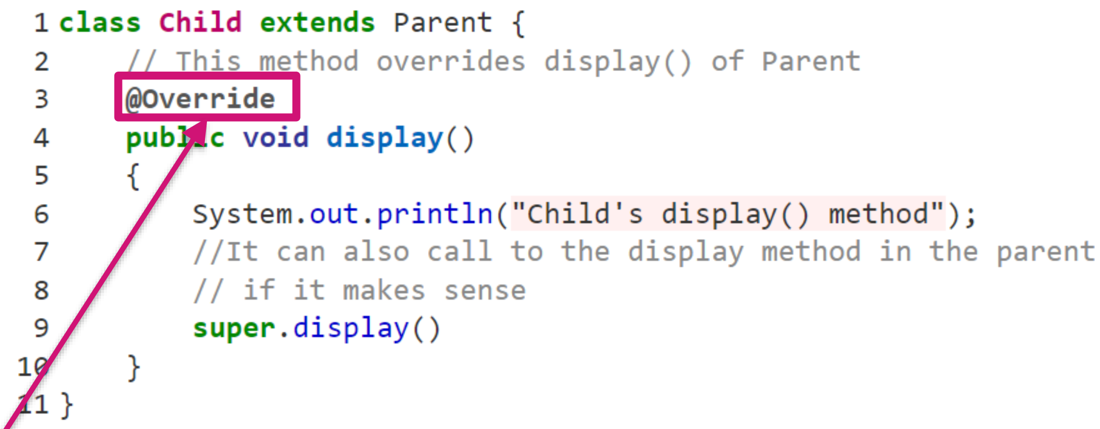
METHOD
OVERRIDING

superclass



```java
class Parent {
    public void display()
    {
        System.out.println("Parent's display() method");
    }
}

```

subclass



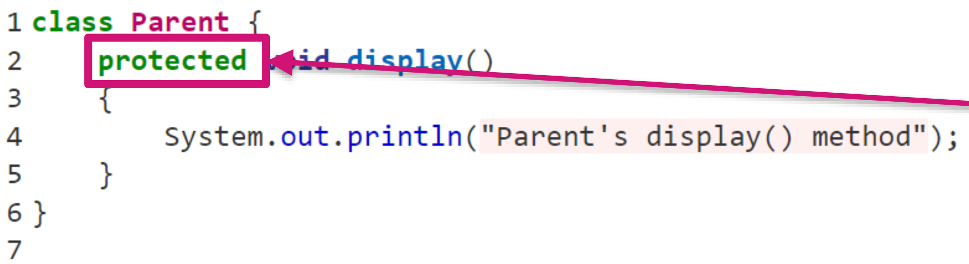
```java
class Child extends Parent {
    // This method overrides display() of Parent
    @Override
    public void display()
    {
        System.out.println("Child's display() method");
        //It can also call to the display method in the parent
        // if it makes sense
        super.display()
    }
}
```

The @override keyword at the top of the method is optional

ACCESS MODIFIERS ANDD
OVERRIDING

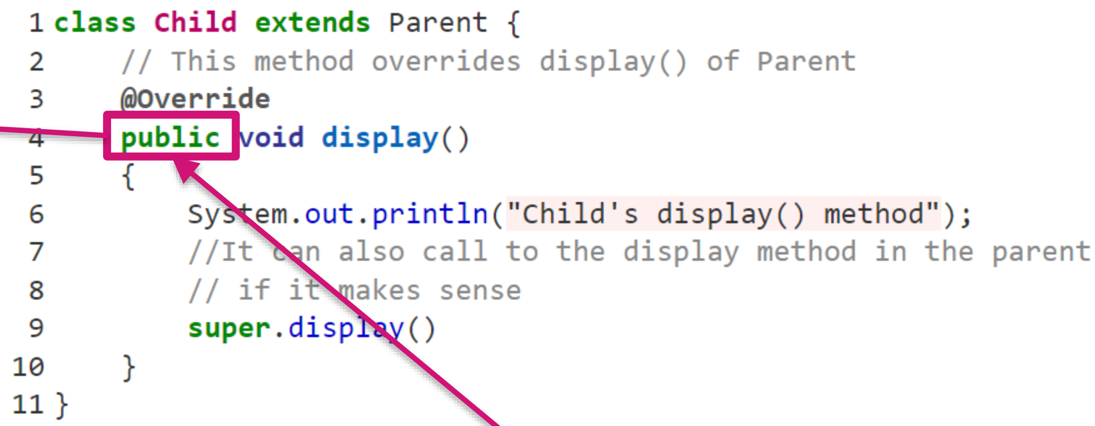
The **access modifier** for an overriding method can allow **more, but not less, access** than the overridden method.

superclass



```
class Parent {
    protected void display()
    {
        System.out.println("Parent's display() method");
    }
}

```

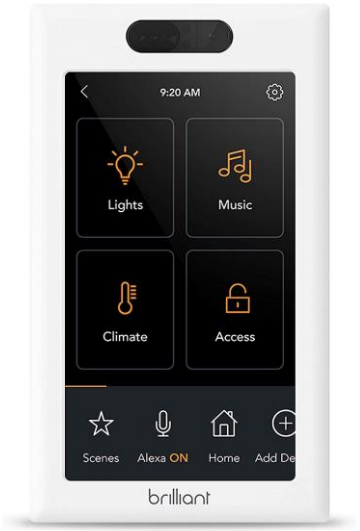
subclass

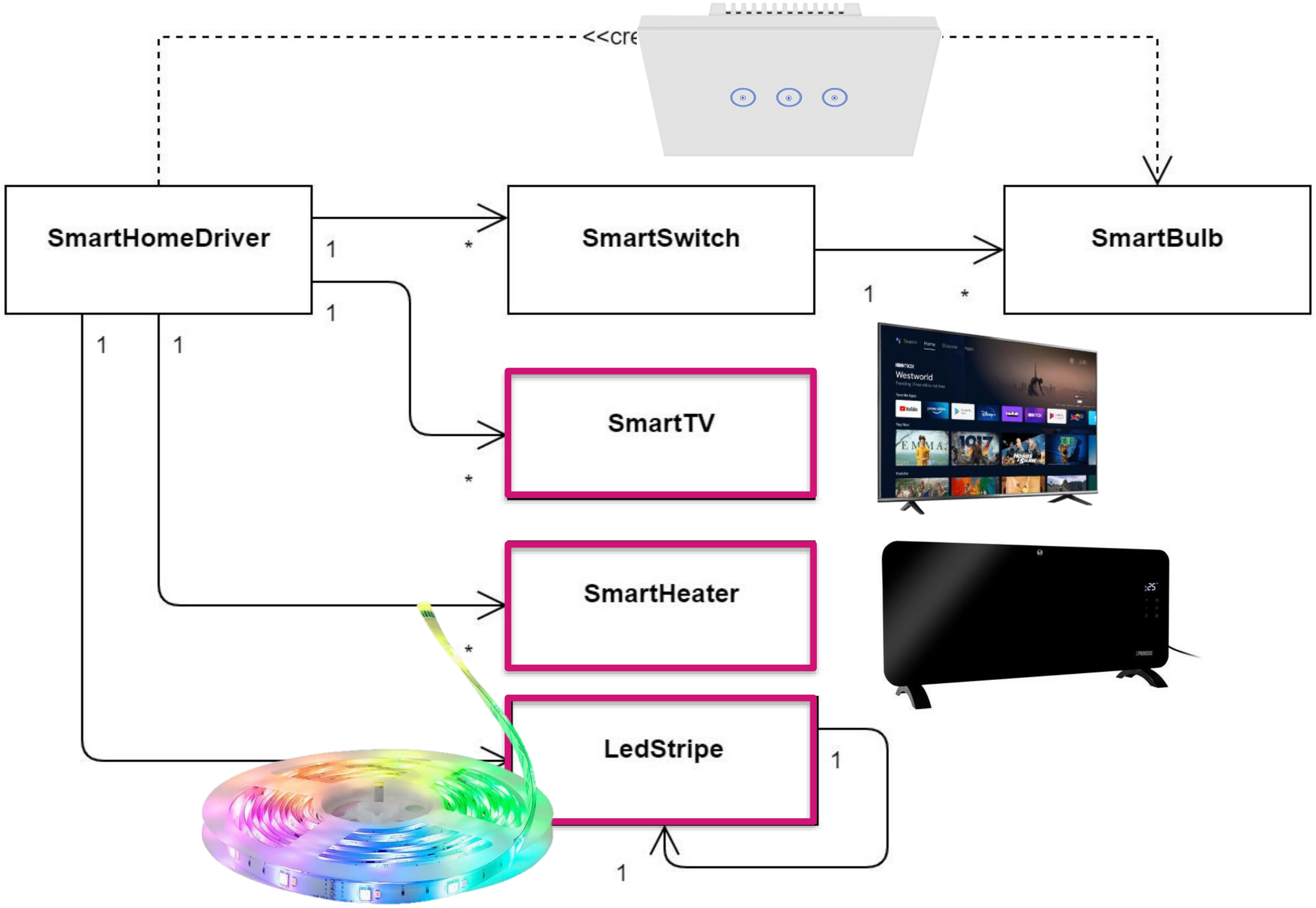


```
class Child extends Parent {
    // This method overrides display() of Parent
    @Override
    public void display()
    {
        System.out.println("Child's display() method");
        //It can also call to the display method in the parent
        // if it makes sense
        super.display()
    }
}
```

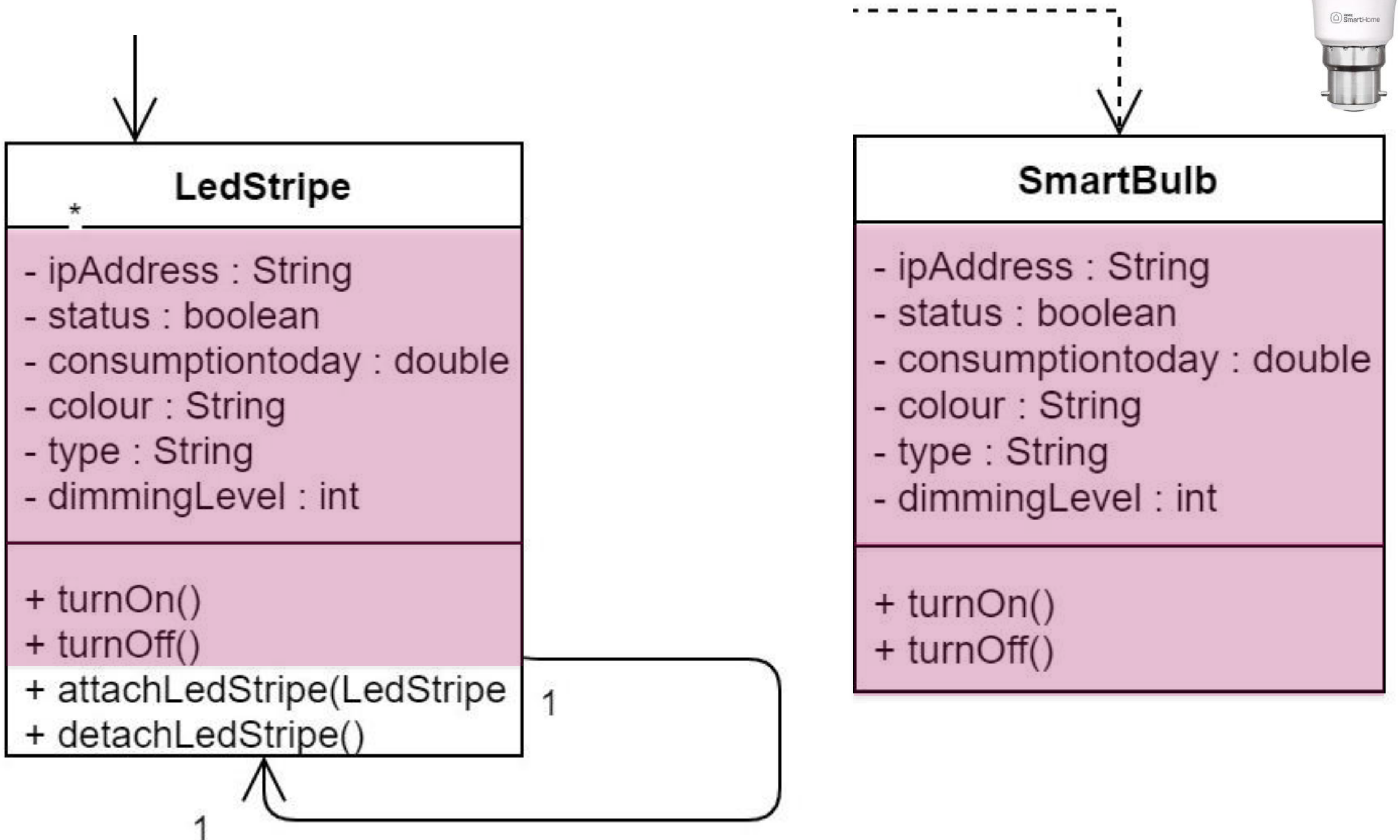
For example, a protected method in the superclass can be made public, but not private, in the subclass.

CAN INHERITANCE BE APPLIED HERE?





DETAILED
CLASS DIAGRAM



LedStripe

*

- ipAddress : String
- status : boolean
- consumptiontoday : double
- colour : String
- type : String
- dimmingLevel : int

+ turnOn()
+ turnOff()
+ attachLedStripe(LedStripe
+ detachLedStripe()

SmartBulb

- ipAddress : String
- status : boolean
- consumptiontoday : double
- colour : String
- type : String
- dimmingLevel : int

+ turnOn()
+ turnOff()

Summary

Inheritance

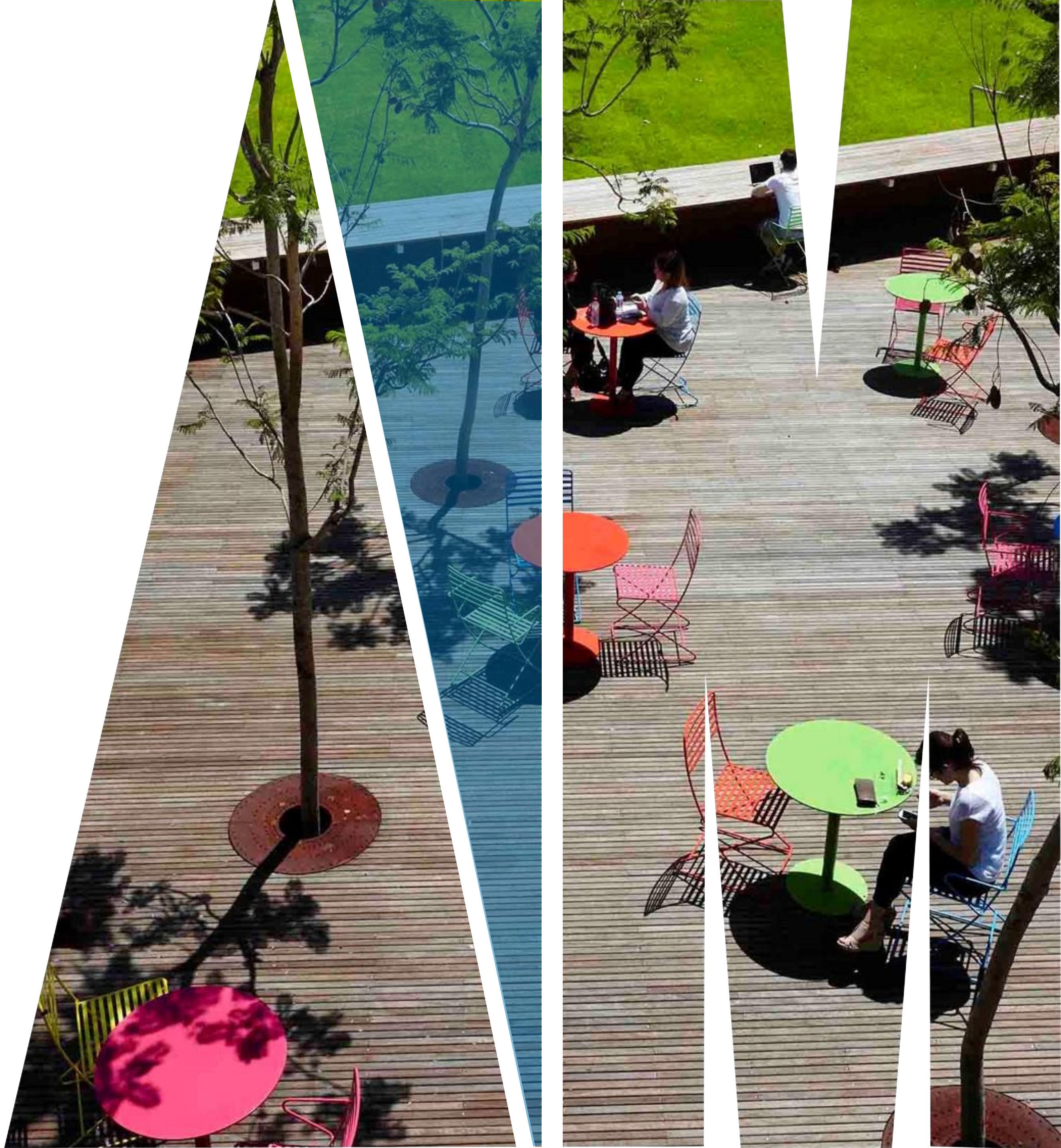
UML representation

Java Syntax

Access modifiers (protected)

MONASH
University

Thanks



MONASH
University