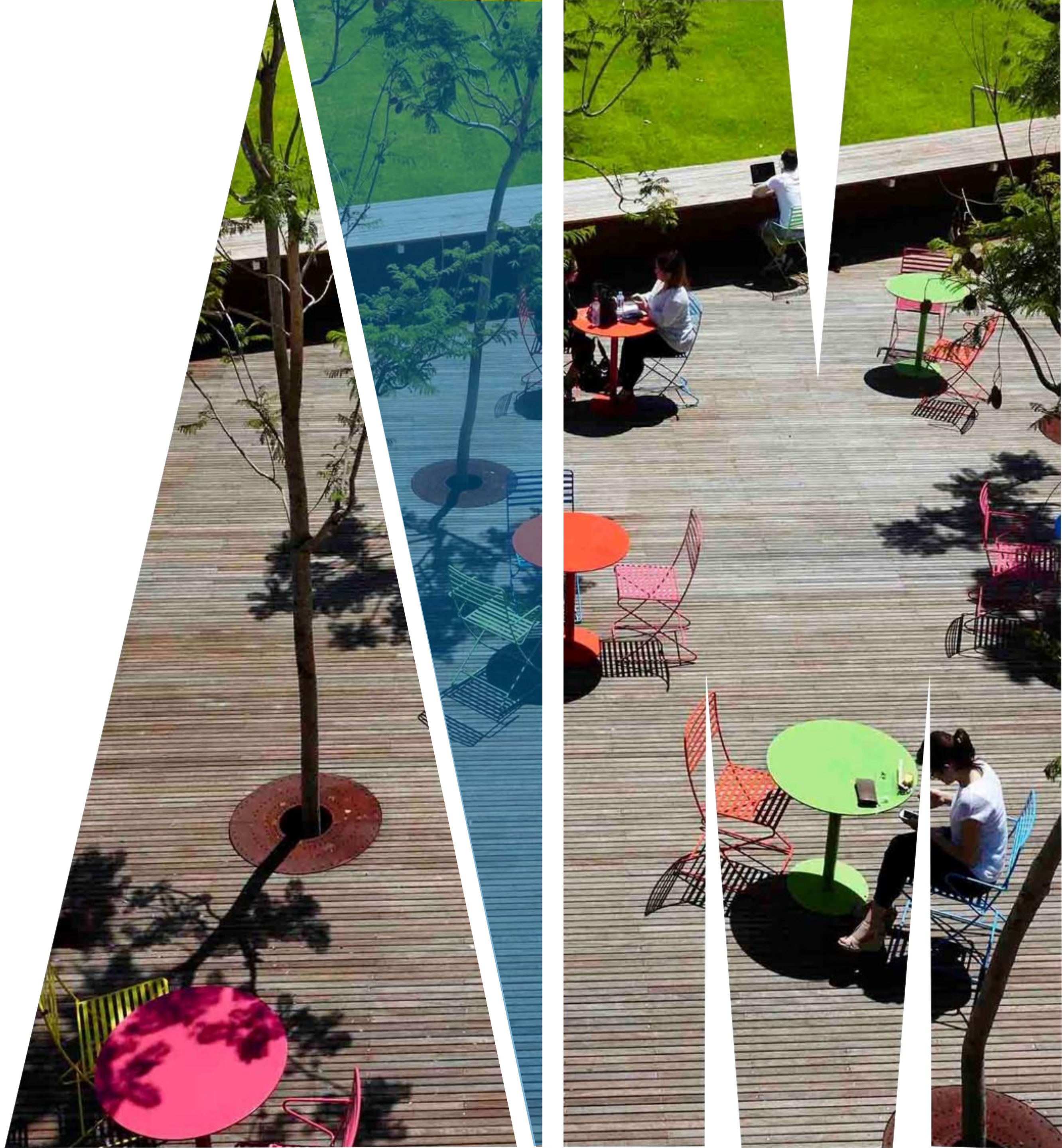
MONASH University

FIT2099 Object-Oriented Design and Implementation

Three core design principles



MONASH University

Outline

Three design principles

Design smells

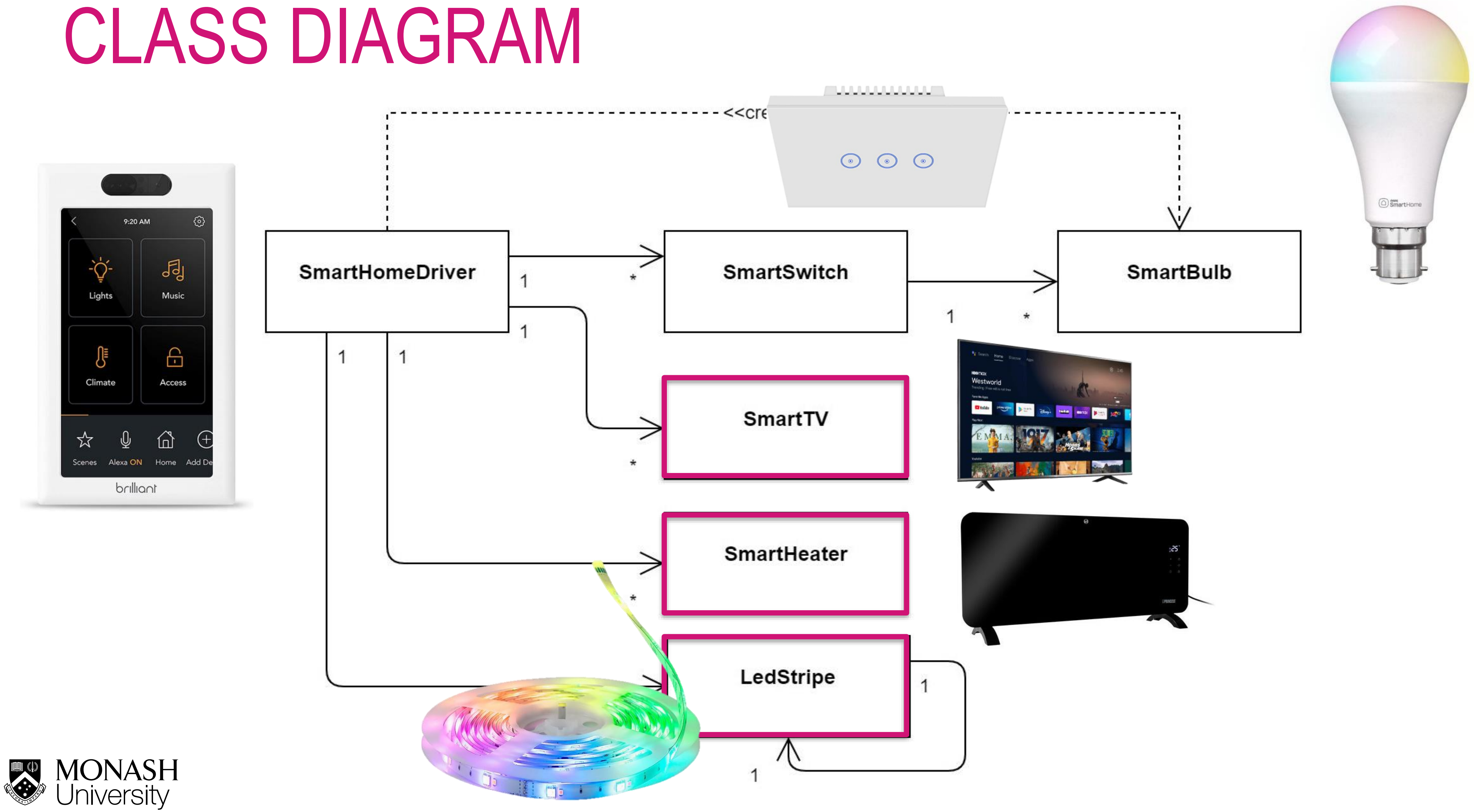
The Smart Home example

BEFORE WE BEGIN…

This and some of the next lectures refer to example code for a Smart Home system. You will find the code that is relevant to each topic on Moodle. Please download it and refer to it as you watch the lecture.

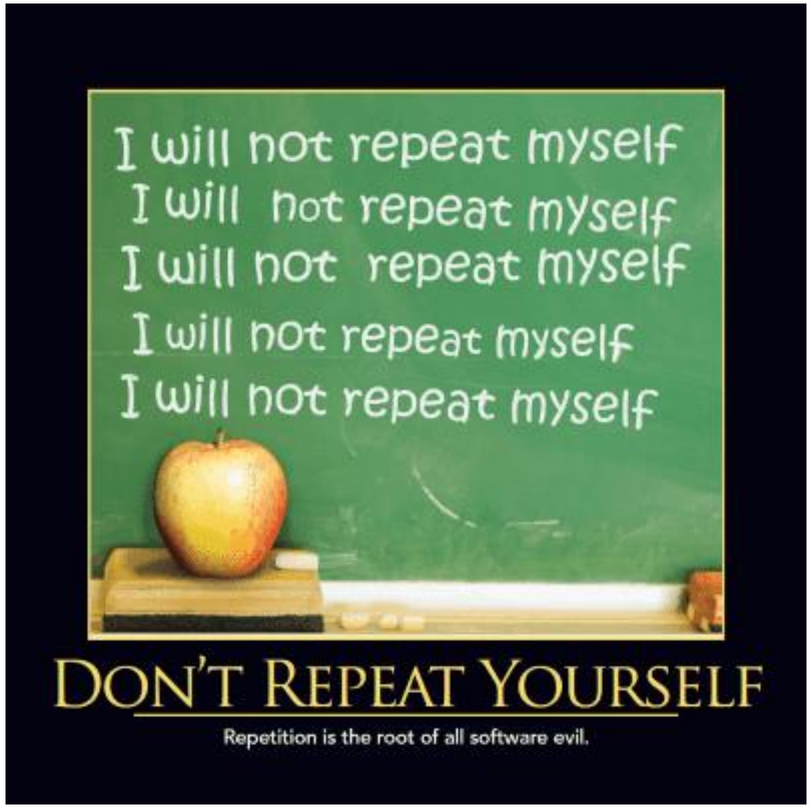
We are going to design a fictional IOT control system for managing Smart home devices.

INITIAL
CLASS DIAGRAM



PRINCIPLE A
DON'T REPEAT YOURSELF (DRY)

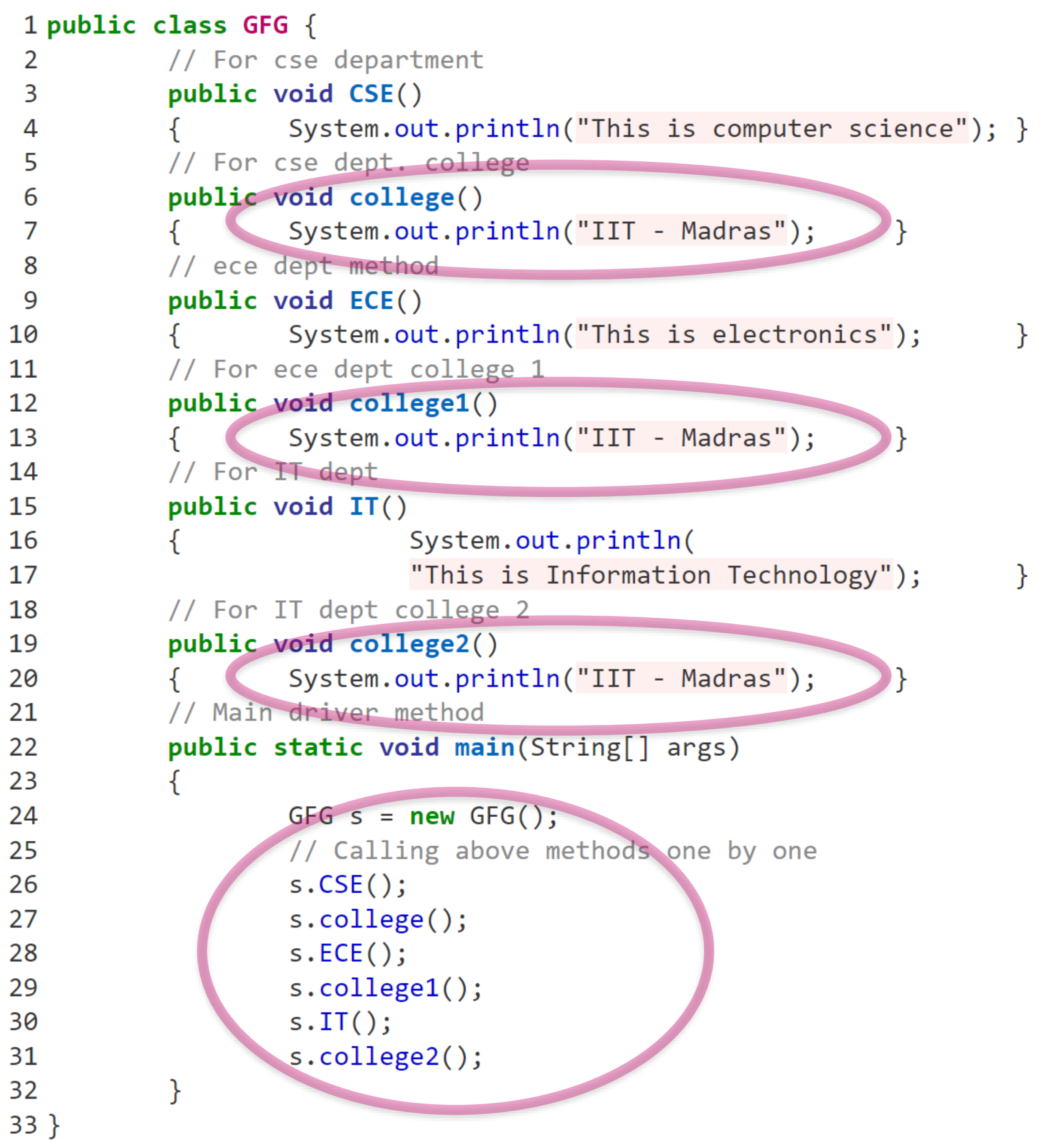
Don't repeat yourself" (DRY) is a principle of software development aimed at reducing repetition of software patterns, replacing repeated code with **abstractions** to avoid redundancy.



PRINCIPLE A

DON’T REPEAT YOURSELF

(“bad” example)



```
public class GFG {
        // For cse department
        public void CSE()
        {       System.out.println("This is computer science"); }
        // For cse dept. college
        public void college()
        {       System.out.println("IIT - Madras");     }
        // ece dept method
        public void ECE()
        {       System.out.println("This is electronics");      }
        // For ece dept college 1
        public void college1()
        {       System.out.println("IIT - Madras");     }
        // For IT dept
        public void IT()
        {               System.out.println(
                        "This is Information Technology");      }
        // For IT dept college 2
        public void college2()
        {       System.out.println("IIT - Madras");     }
        // Main driver method
        public static void main(String[] args)
        {
                GFG s = new GFG();
                // Calling above methods one by one
                s.CSE();
                s.college();
                s.ECE();
                s.college1();
                s.IT();
                s.college2();
        }
}
```

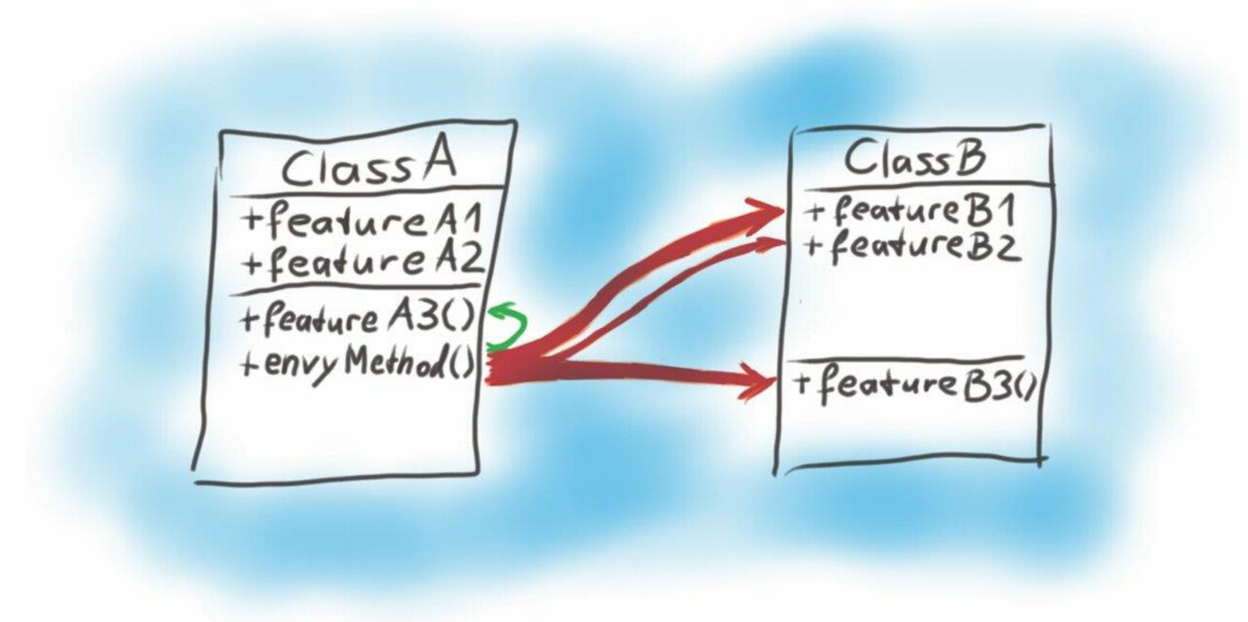
MONASH University

PRINCIPLE B

CLASSES SHOULD BE RESPONSIBLE FOR THEIR OWN PROPERTIES

As a basic rule, if things change at the same time, you should keep them in the same place.

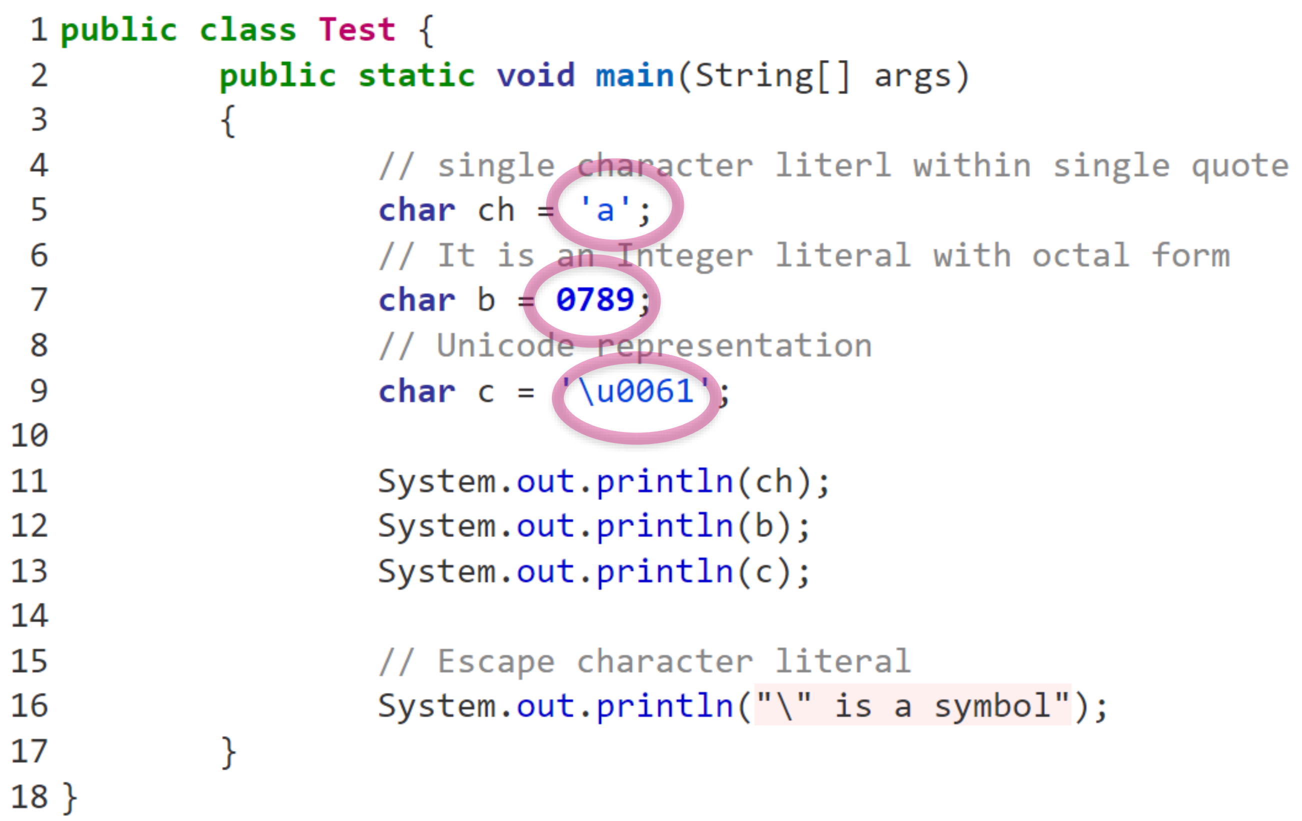
Note: this is related to a design smell called “feature envy” and a principle called “single-responsibility (SRP). We will more deeply cover these later.



PRINCIPLE C
AVOID EXCESIVE USE OF LITERALS

Every piece of software contains **literals** (usually numbers, strings or booleans).

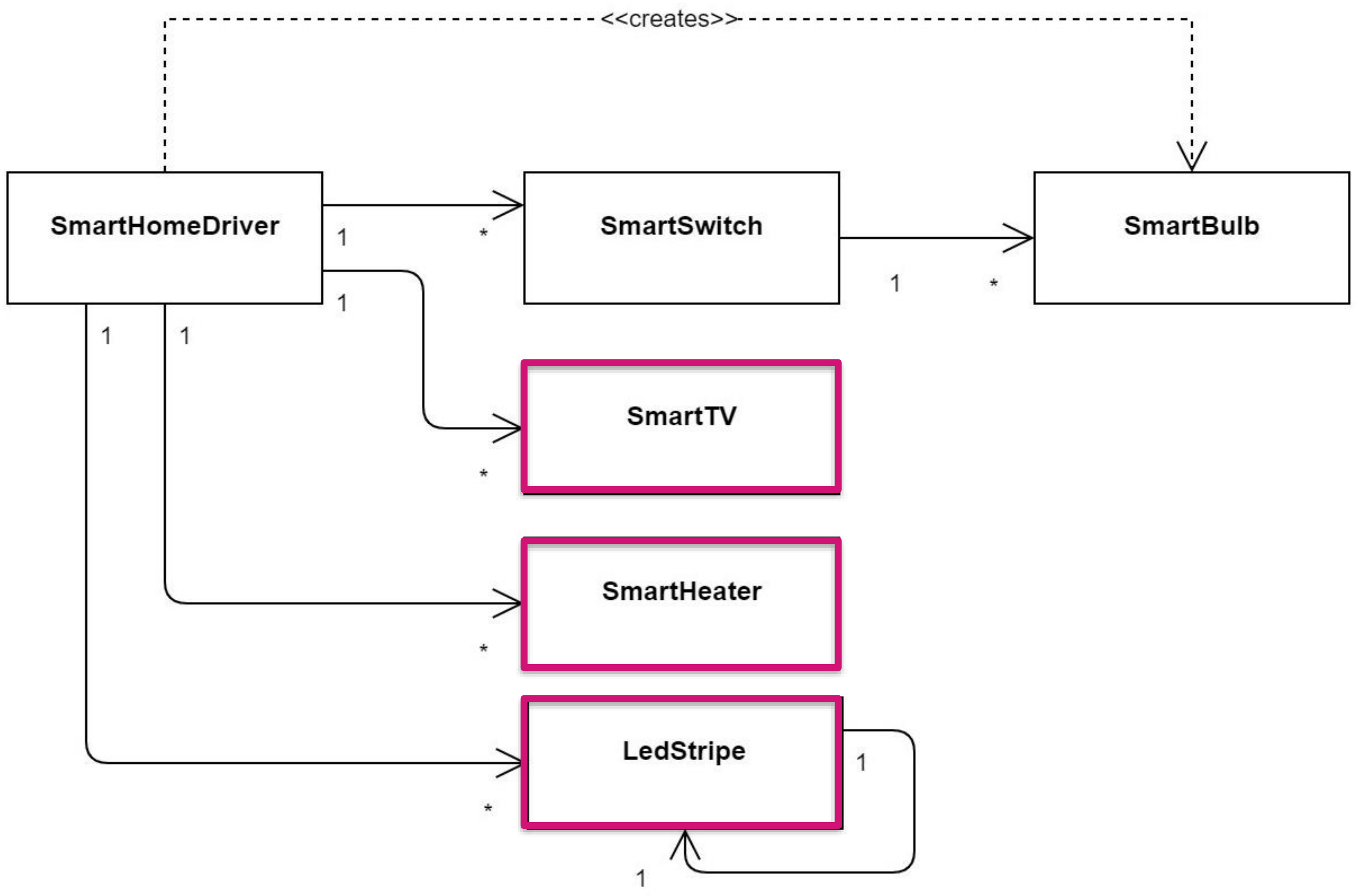
These are **fixed values in source code**, commonly related to application configuration, parts of the business logic, natural or language constants, etc..



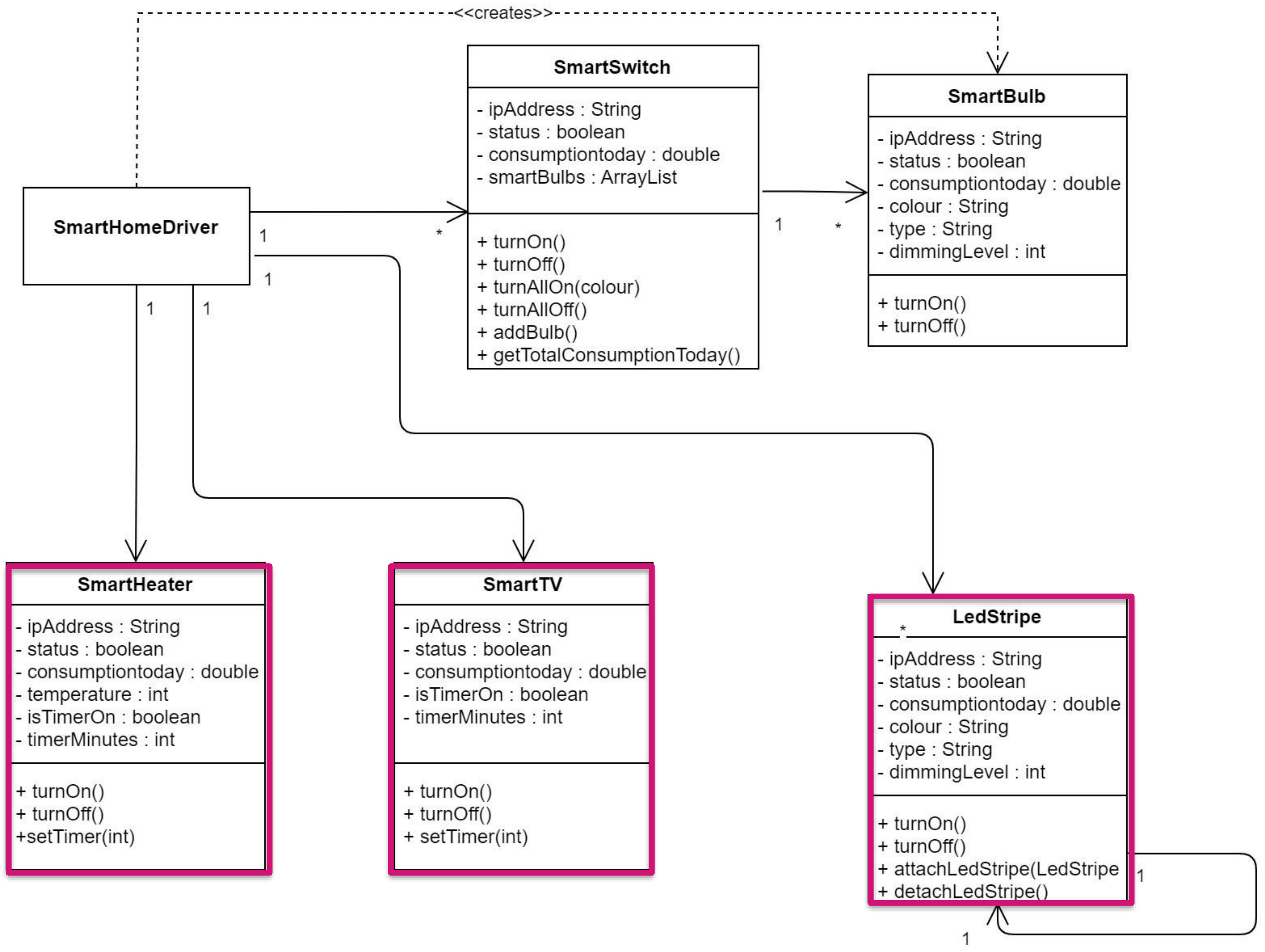
```java
public class Test {
        public static void main(String[] args)
        {
                // single character literl within single quote
                char ch = 'a';
                // It is an Integer literal with octal form
                char b = 0789;
                // Unicode representation
                char c = '\u0061';

                System.out.println(ch);
                System.out.println(b);
                System.out.println(c);

                // Escape character literal
                System.out.println("\" is a symbol");
        }
}
```

MONASH University

INITIAL
CLASS DIAGRAM

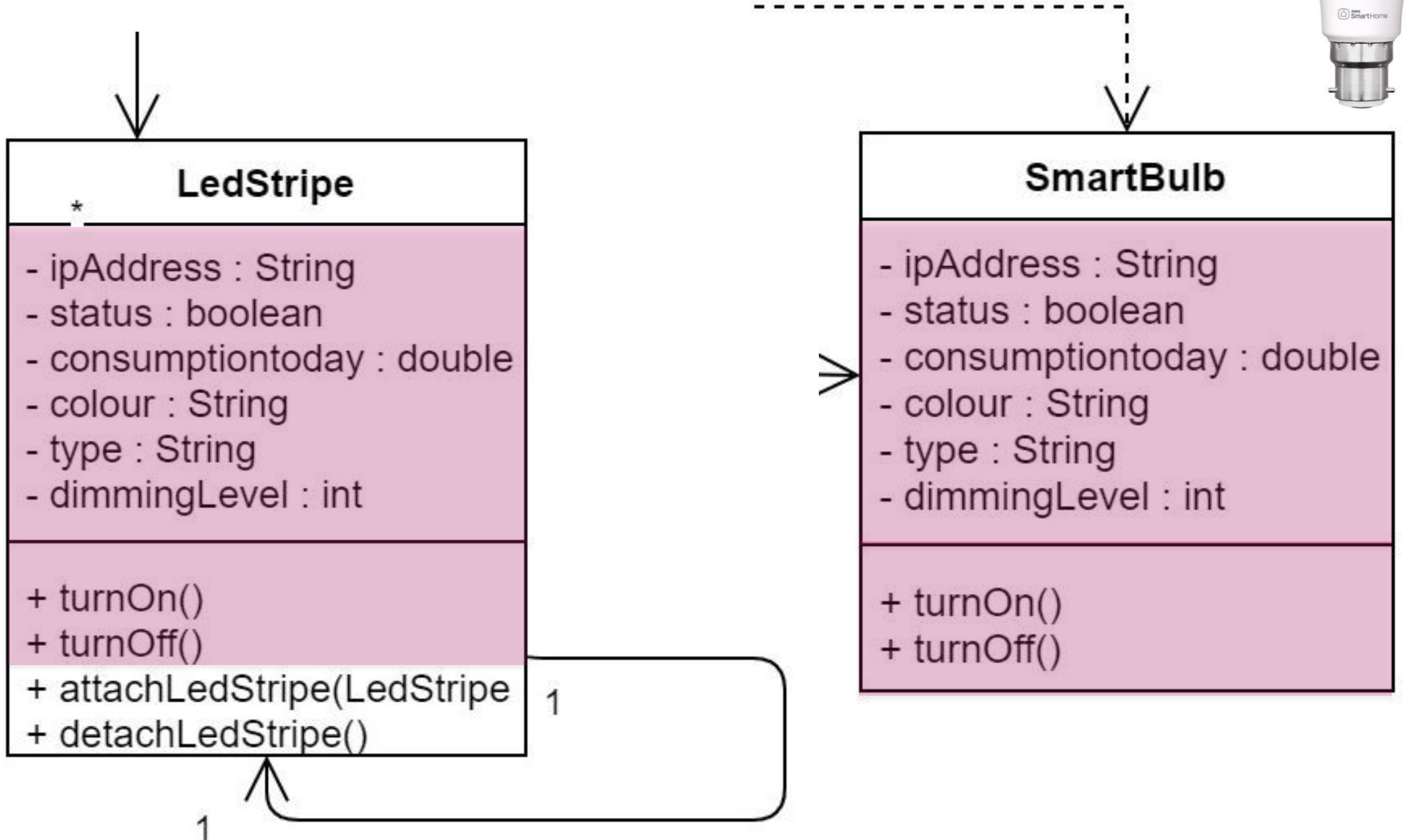


DETAILED CLASS DIAGRAM



MONASH University

DETAILED CLASS DIAGRAM



Summary

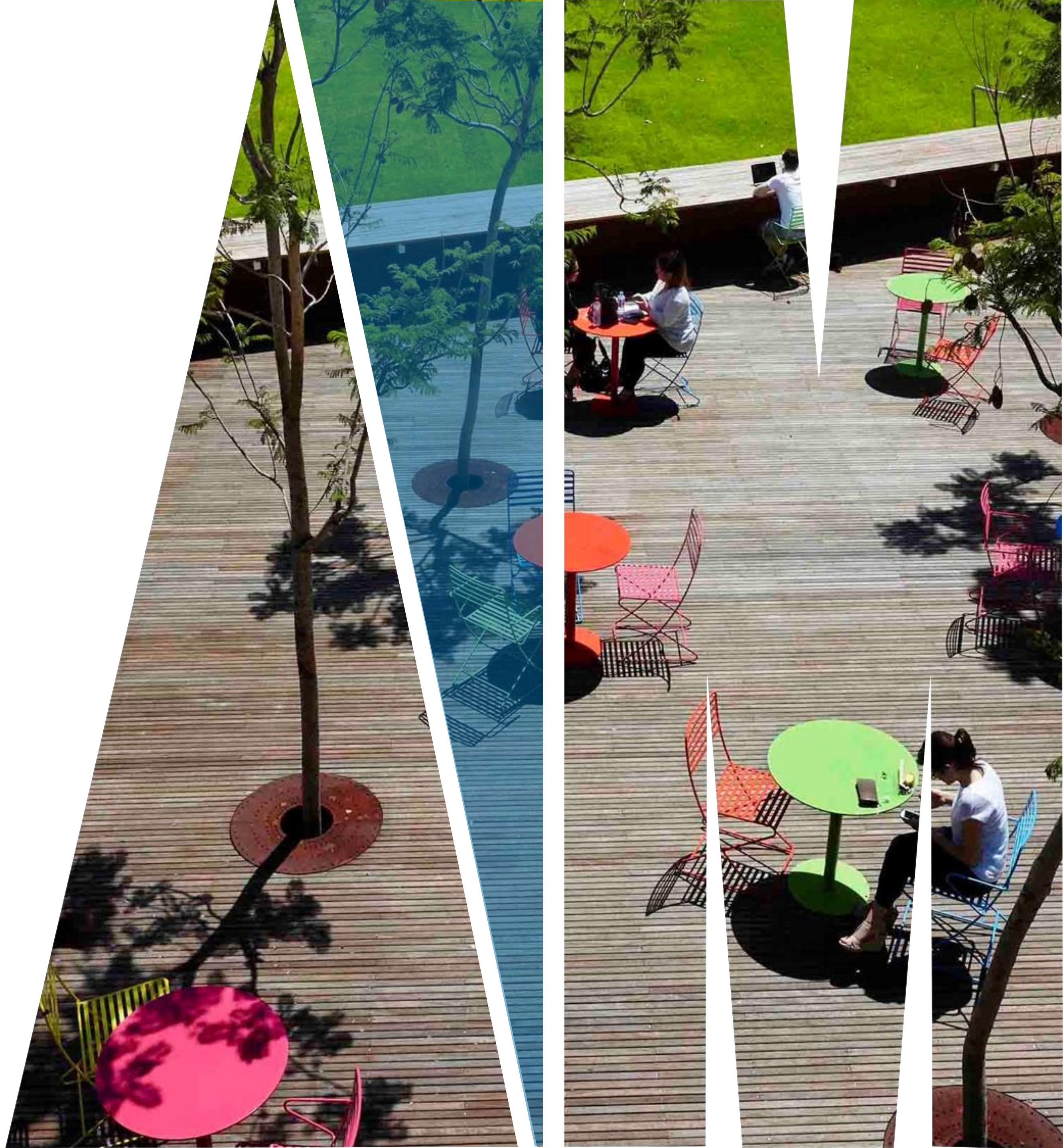
Three design principles

Design smells

The Smart Home example

MONASH University

Thanks



MONASH University