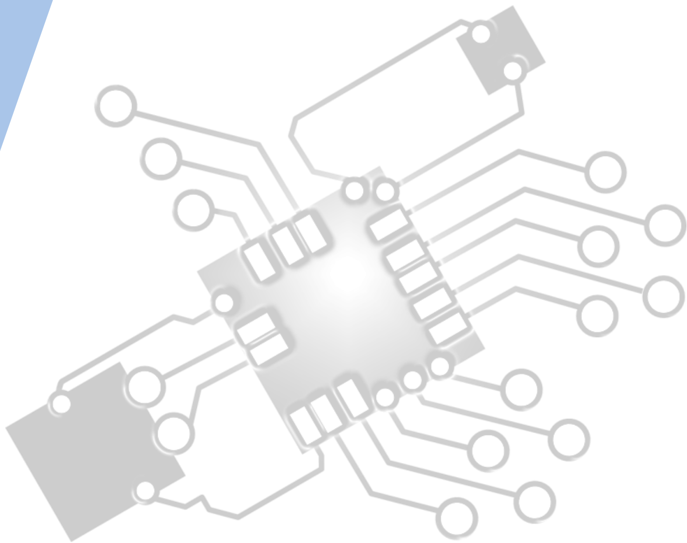
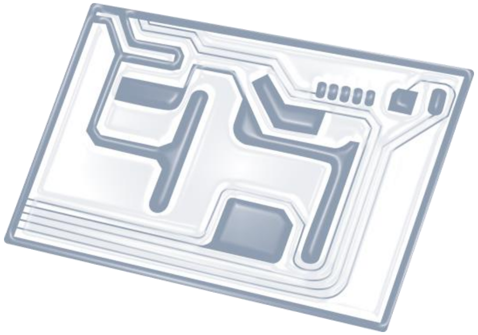
Features of OOP

IB Computer Science



Content developed by
Dartford Grammar School
Computer Science Department

HL Topics 1-7, D1-4



1: System design

2: Computer Organisation

3: Networks

4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

D: OOP

HL & SL D.2 Overview

D.2 Features of OOP

D.2.1 Define the term encapsulation

D.2.2 Define the term inheritance

D.2.3 Define the term polymorphism

D.2.4 Explain the advantages of encapsulation

D.2.5 Explain the advantages of inheritance

D.2.6 Explain the advantages of polymorphism

D.2.7 Describe the advantages of libraries of objects

D.2.8 Describe the disadvantages of OOP

D.2.9 Discuss the use of programming teams

D.2.10 Explain the advantages of modularity in program development

1: System design

2: Computer Organisation

3: Networks

4: Computational thinking

5: Abstract data structures

6: Resource management

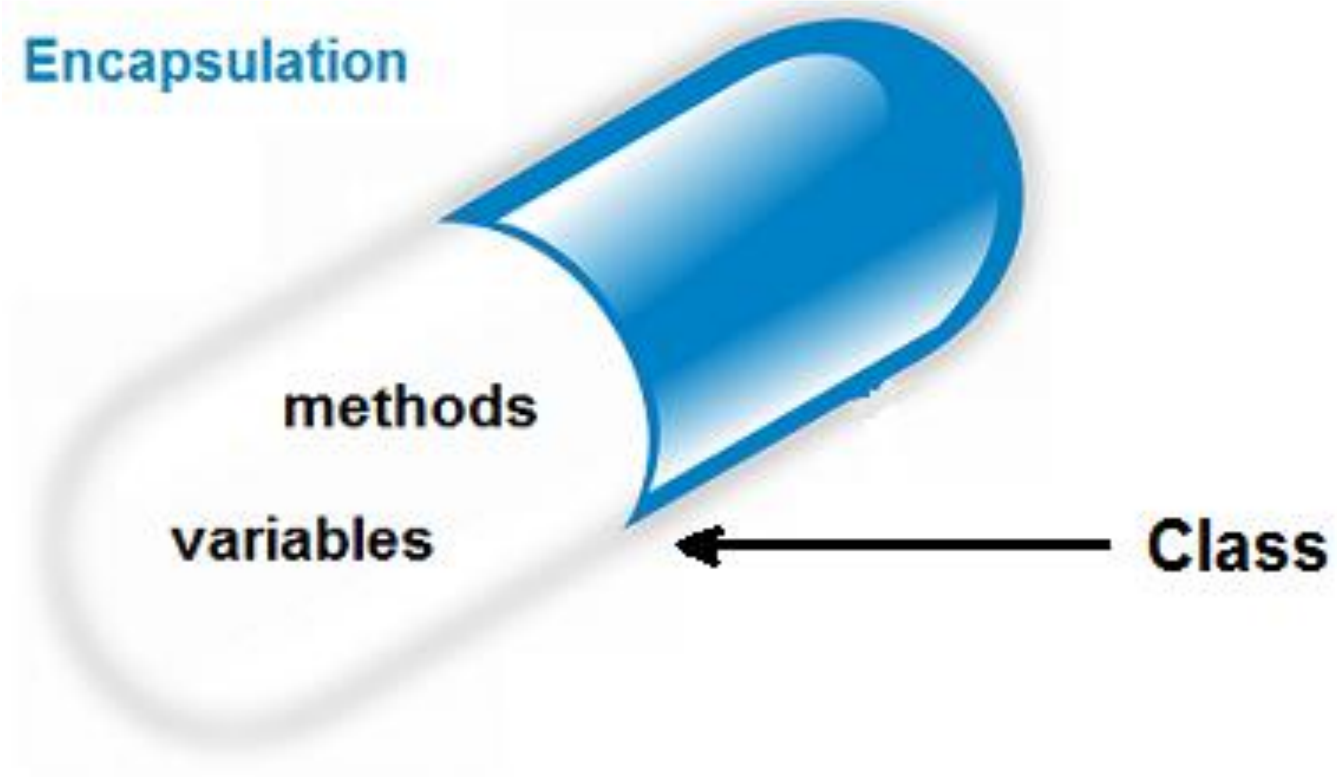
7: Control

D: OOP

Content developed by Dartford Grammar School Computer Science Department

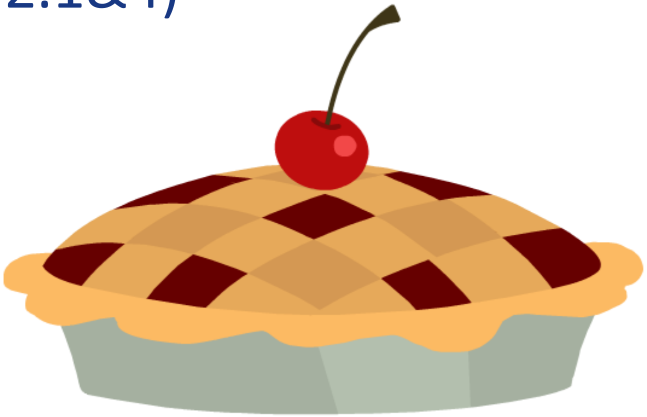
Topic D.2.1

Define the term: **encapsulation**



Four OOP fundamentals:

- **A**bstraction (*See* Topic 4.1.17-20)
- **P**olymorphism (*See* Topic D.2.3&6)
- **I**nheritance (*See* Topic D.2.2&5)
- **E**ncapsulation (*See* Topic D.2.1&4)

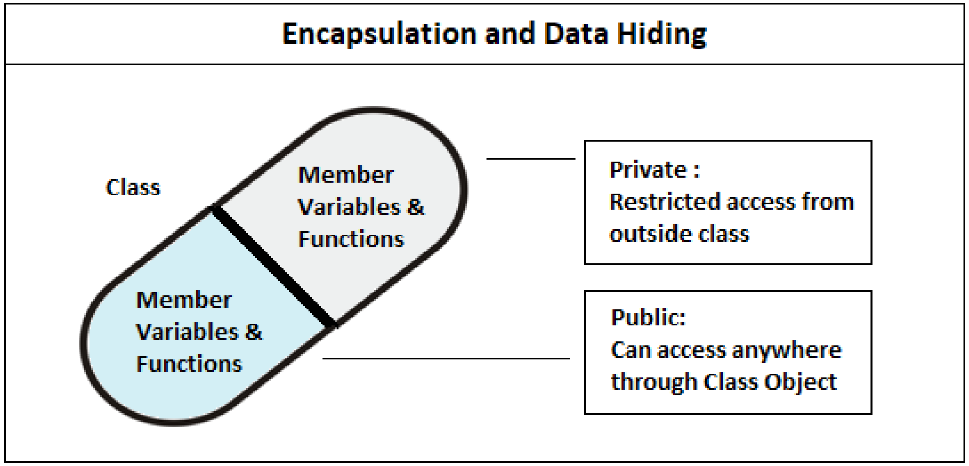


Definition: Encapsulation

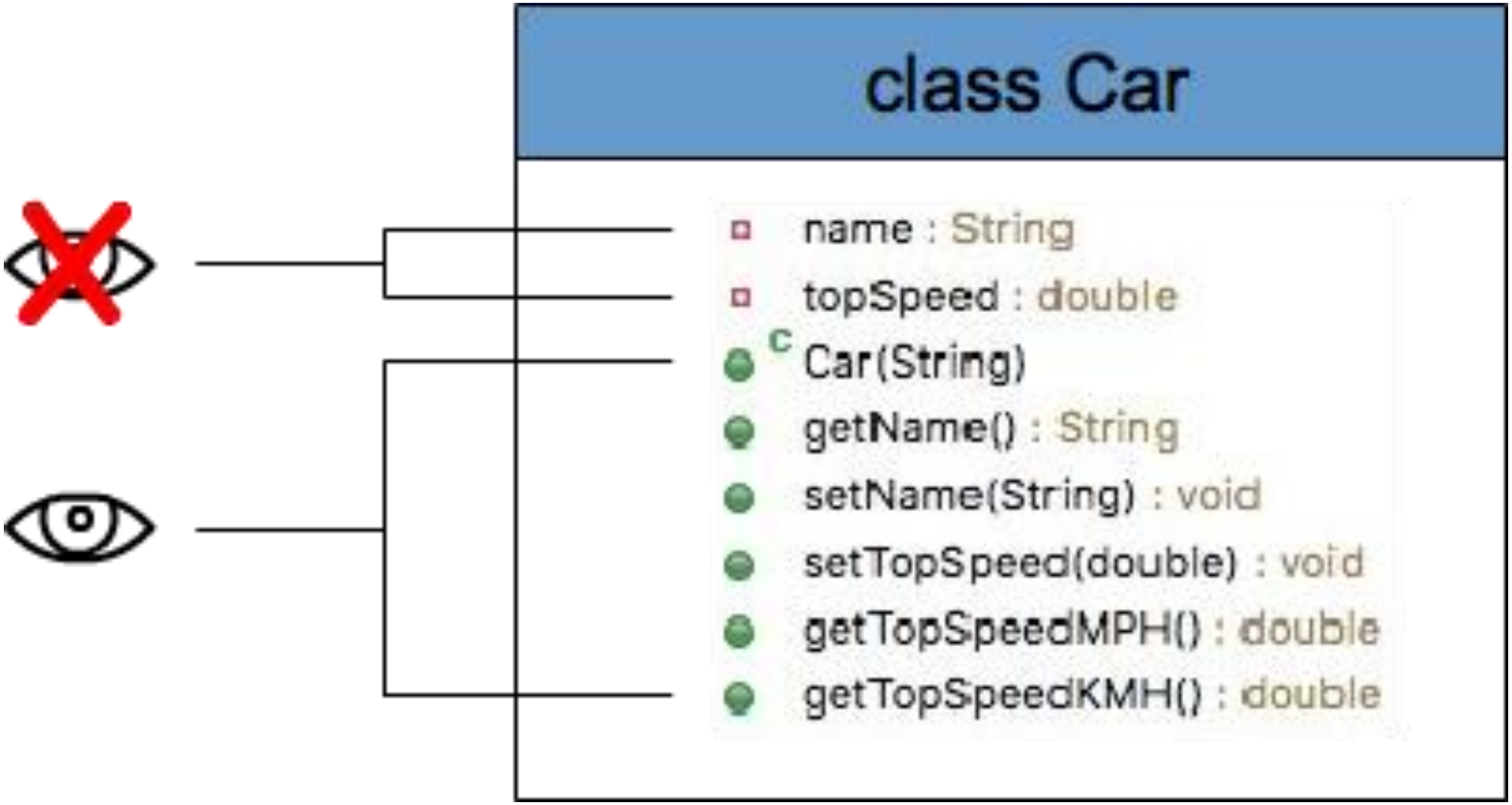
- Encapsulation is the technique of making the **states** in a class **private** and providing access to those states via **public behaviours (methods)**.
- **In short**: **data and actions are limited to the object in which they are created**

Encapsulation = Data Hiding

- If a state is declared **private**, it **cannot be accessed by any method outside the class**, thereby hiding the states (and their contents) within the class.
- For this reason, encapsulation is also referred to as **data hiding**.


Encapsulation and Data Hiding
Class
Member Variables & Functions
Member Variables & Functions
Private : Restricted access from outside class
Public: Can access anywhere through Class Object

Example: UML



class Car

□ name : String

□ topSpeed : double

● ^{c} Car(String)

● getName() : String

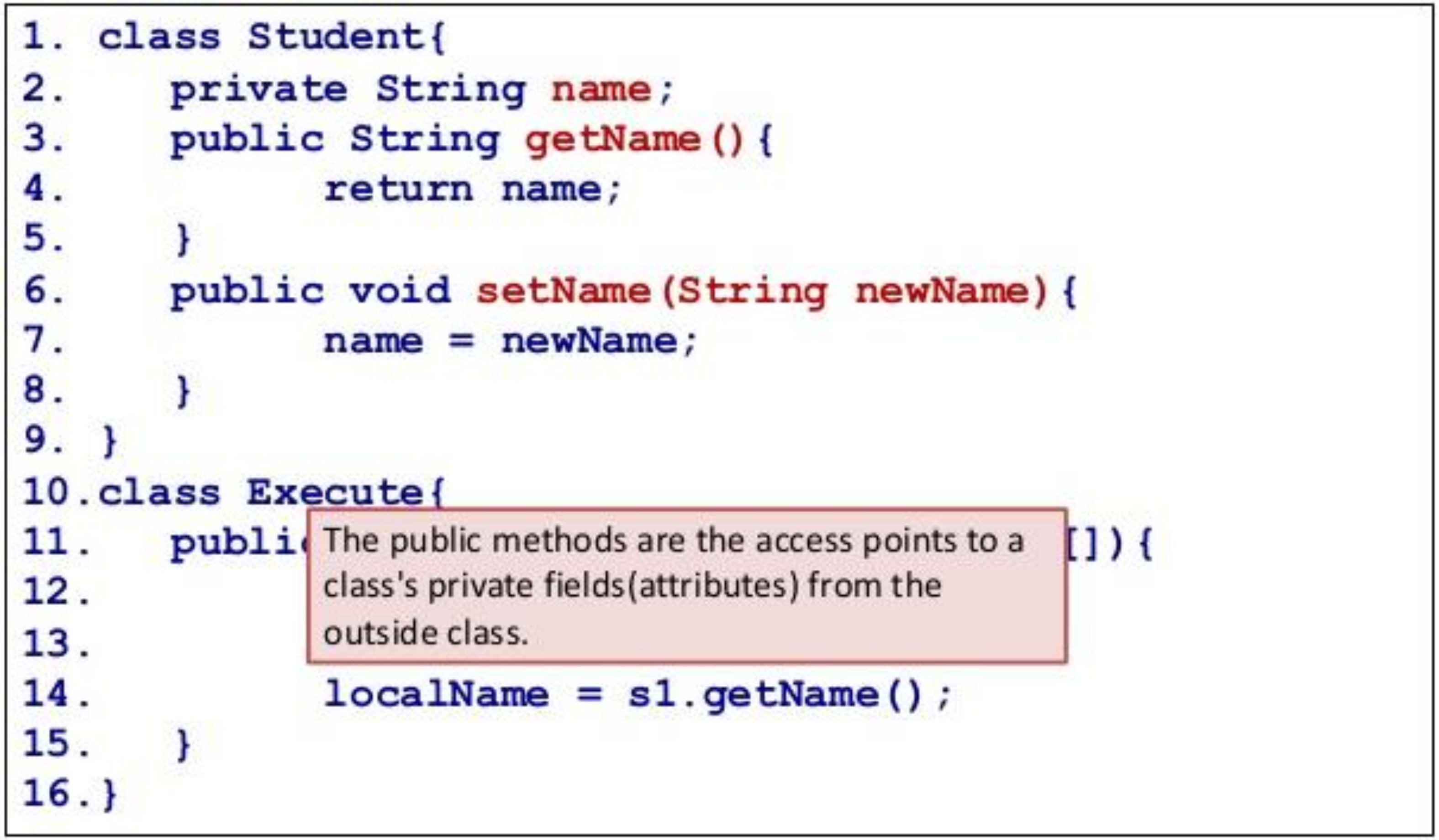
● setName(String) : void

● setTopSpeed(double) : void

● getTopSpeedMPH() : double

● getTopSpeedKMH() : double

Example: Java

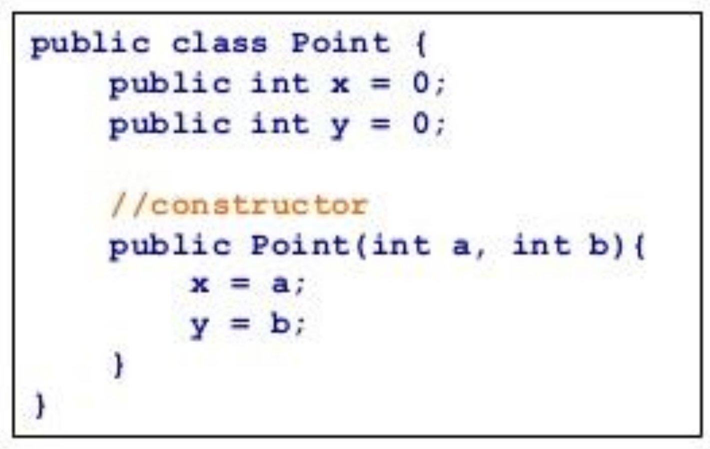


```
class Student{
     private String name;
     public String getName(){
           return name;
     }
     public void setName(String newName){
           name = newName;
     }
}
class Execute{
     publi                                    []){


           localName = s1.getName();
     }
}
```

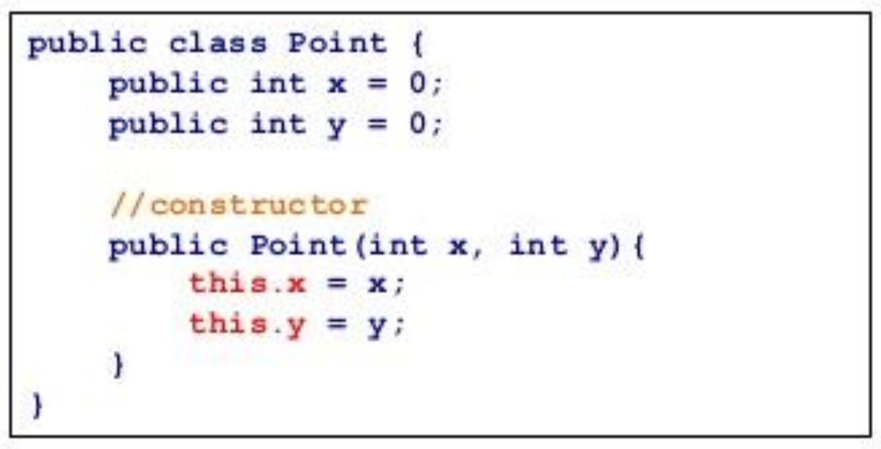
The public methods are the access points to a class's private fields(attributes) from the outside class.

Side note: Java keyword **`this`**

Within a method/constructor, **`this`** is a reference to the **current object** (the object whose method/constructor is being called)

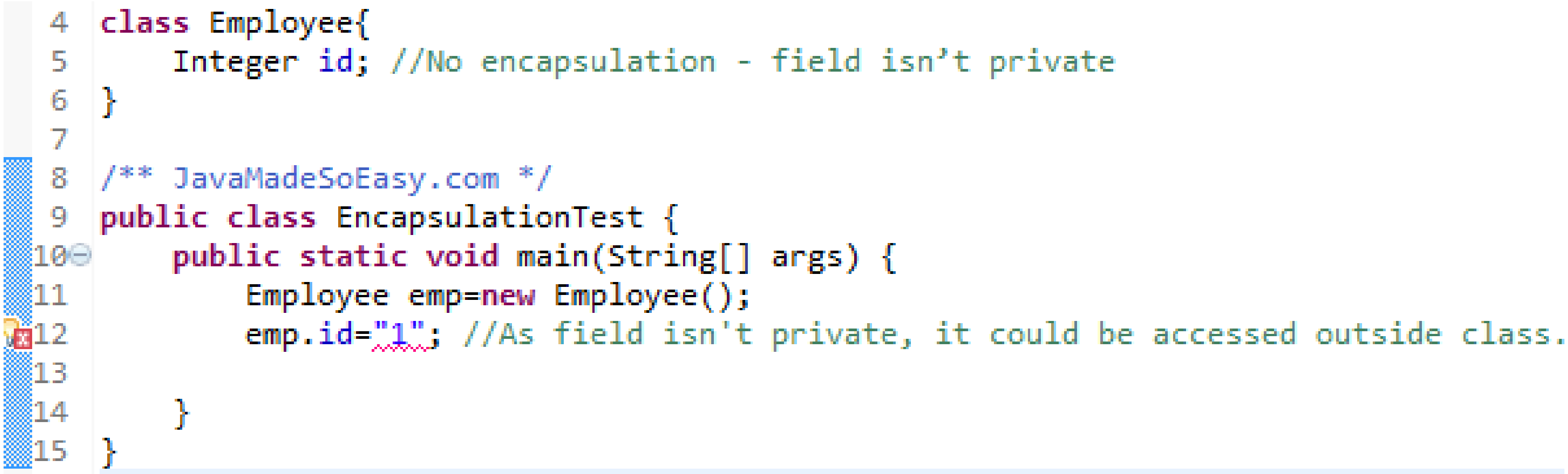


```
public class Point {
     public int x = 0;
     public int y = 0;

     //constructor
     public Point(int a, int b){
          x = a;
          y = b;
     }
}
```



```
public class Point {
     public int x = 0;
     public int y = 0;

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

Example of non-encapsulation: Java



```
class Employee{
    Integer id; //No encapsulation - field isn’t private
}

/** JavaMadeSoEasy.com */
public class EncapsulationTest {
    public static void main(String[] args) {
        Employee emp=new Employee();
        emp.id="1"; //As field isn't private, it could be accessed outside class.

    }
}
```

This is potentially very dangerous as methods outside the class can directly change an object’s state values.