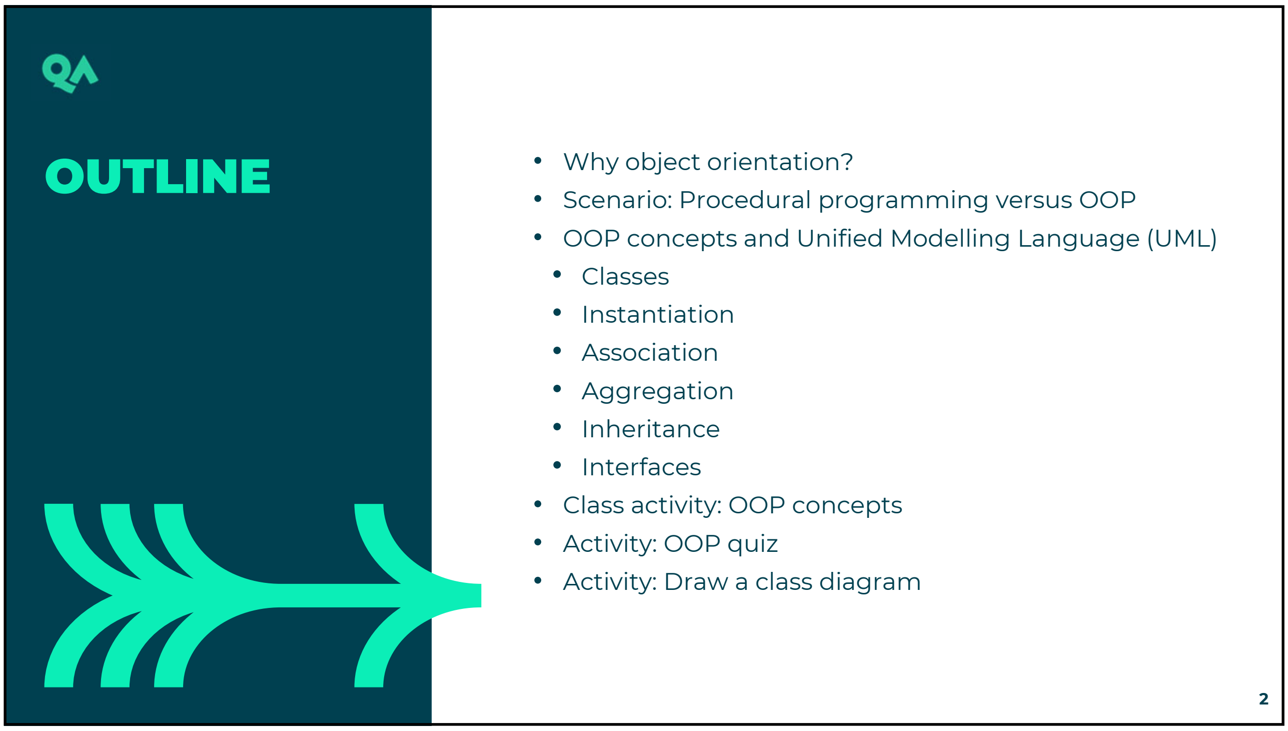
Object Oriented Programming (OOP)

1

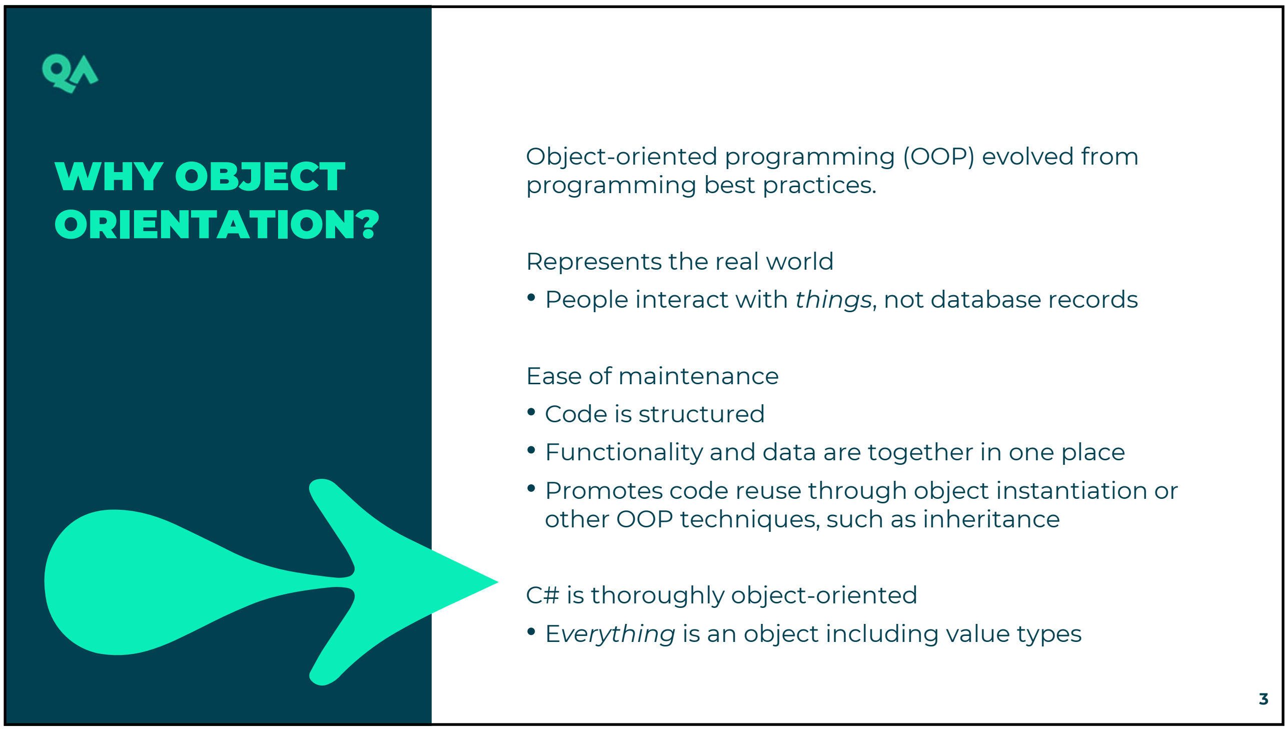


OUTLINE

- Why object orientation?
- Scenario: Procedural programming versus OOP
- OOP concepts and Unified Modelling Language (UML)
  - Classes
  - Instantiation
  - Association
  - Aggregation
  - Inheritance
  - Interfaces
- Class activity: OOP concepts
- Activity: OOP quiz
- Activity: Draw a class diagram

2

2



WHY OBJECT ORIENTATION?

Object-oriented programming (OOP) evolved from programming best practices.

Represents the real world

- People interact with *things*, not database records

Ease of maintenance

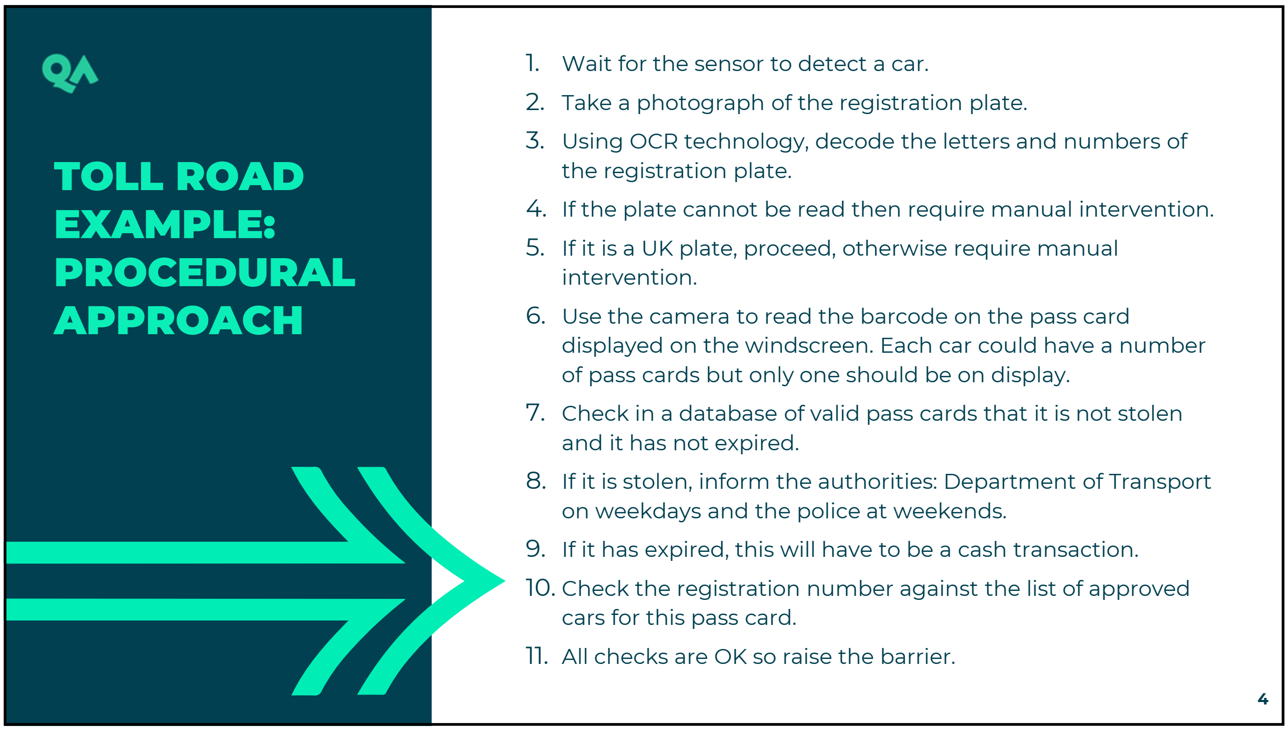
- Code is structured
- Functionality and data are together in one place
- Promotes code reuse through object instantiation or other OOP techniques, such as inheritance

C# is thoroughly object-oriented

- *Everything* is an object including value types

3

3



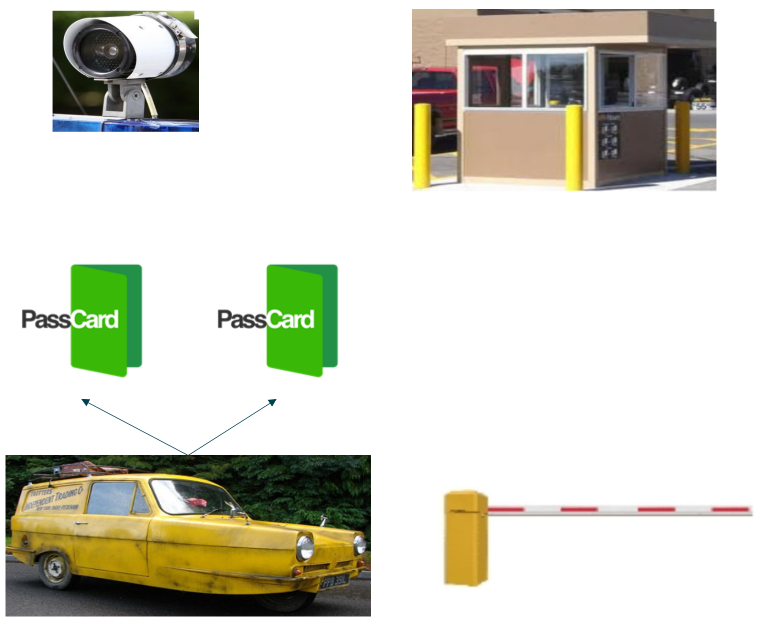
TOLL ROAD EXAMPLE: PROCEDURAL APPROACH

1. Wait for the sensor to detect a car.
2. Take a photograph of the registration plate.
3. Using OCR technology, decode the letters and numbers of the registration plate.
4. If the plate cannot be read then require manual intervention.
5. If it is a UK plate, proceed, otherwise require manual intervention.
6. Use the camera to read the barcode on the pass card displayed on the windscreen. Each car could have a number of pass cards but only one should be on display.
7. Check in a database of valid pass cards that it is not stolen and it has not expired.
8. If it is stolen, inform the authorities: Department of Transport on weekdays and the police at weekends.
9. If it has expired, this will have to be a cash transaction.
10. Check the registration number against the list of approved cars for this pass card.
11. All checks are OK so raise the barrier.

4

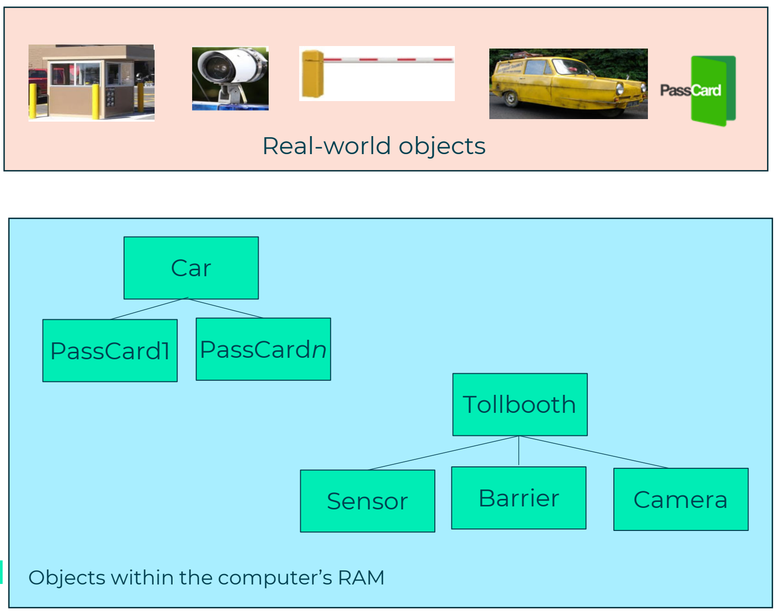
4

TOLL ROAD EXAMPLE: OBJECT ORIENTED APPROACH

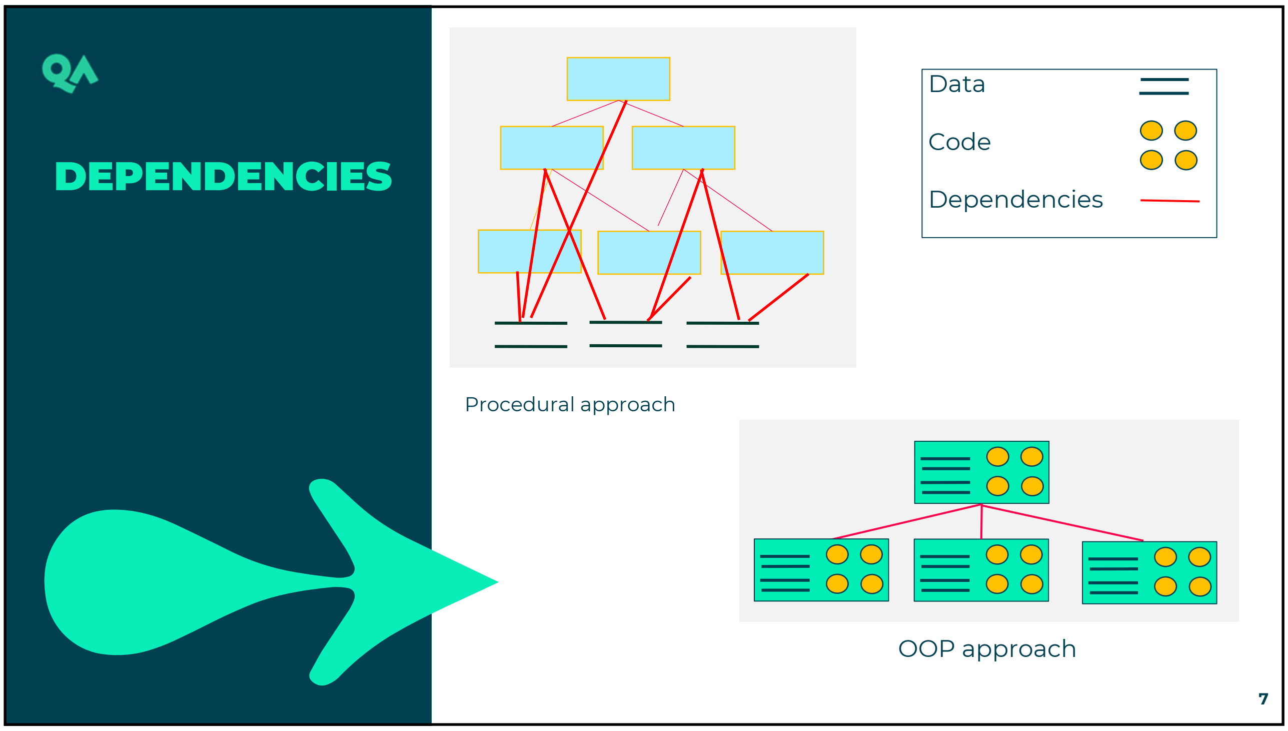


5

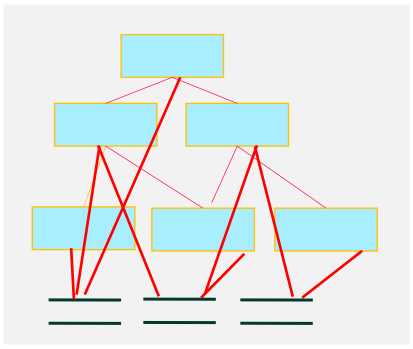
OOP APPROACH



6

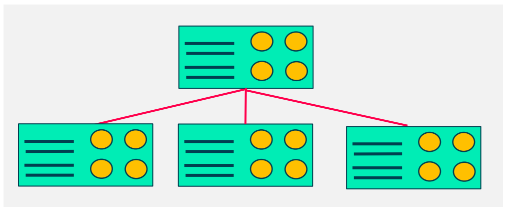


DEPENDENCIES



Procedural approach

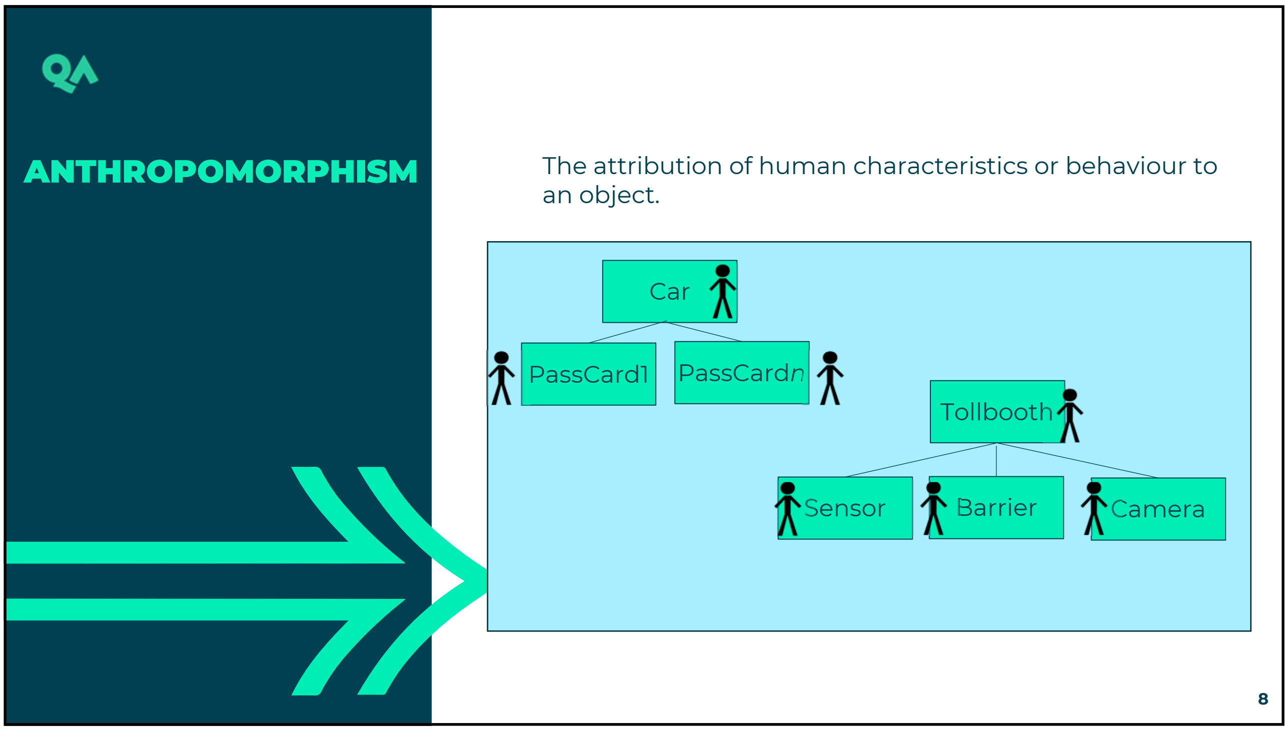
Data	☰
Code	●● ●●
Dependencies	―



OOP approach

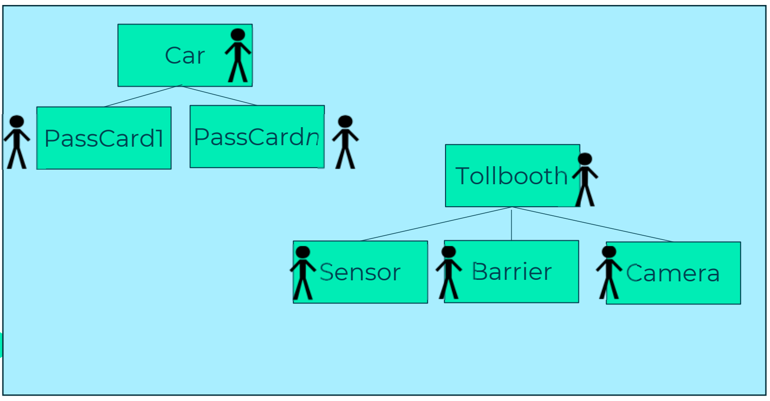
7

7



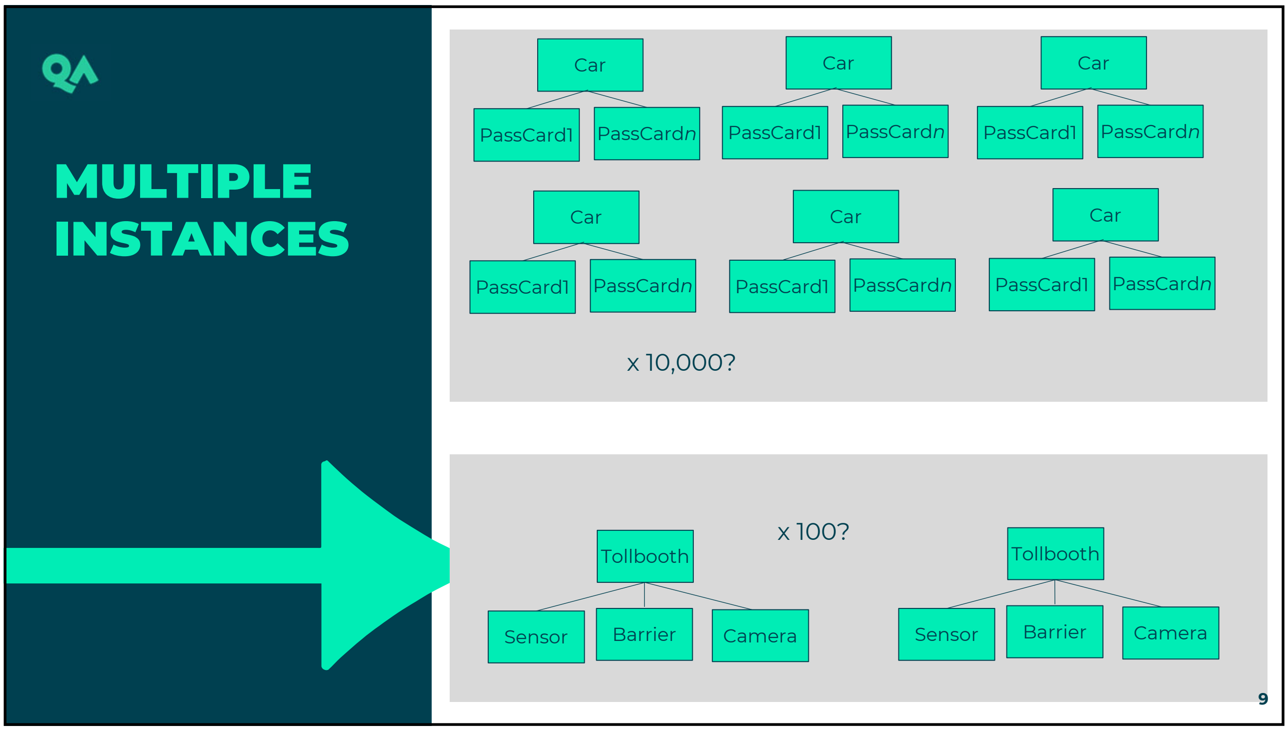
ANTHROPOMORPHISM

The attribution of human characteristics or behaviour to an object.



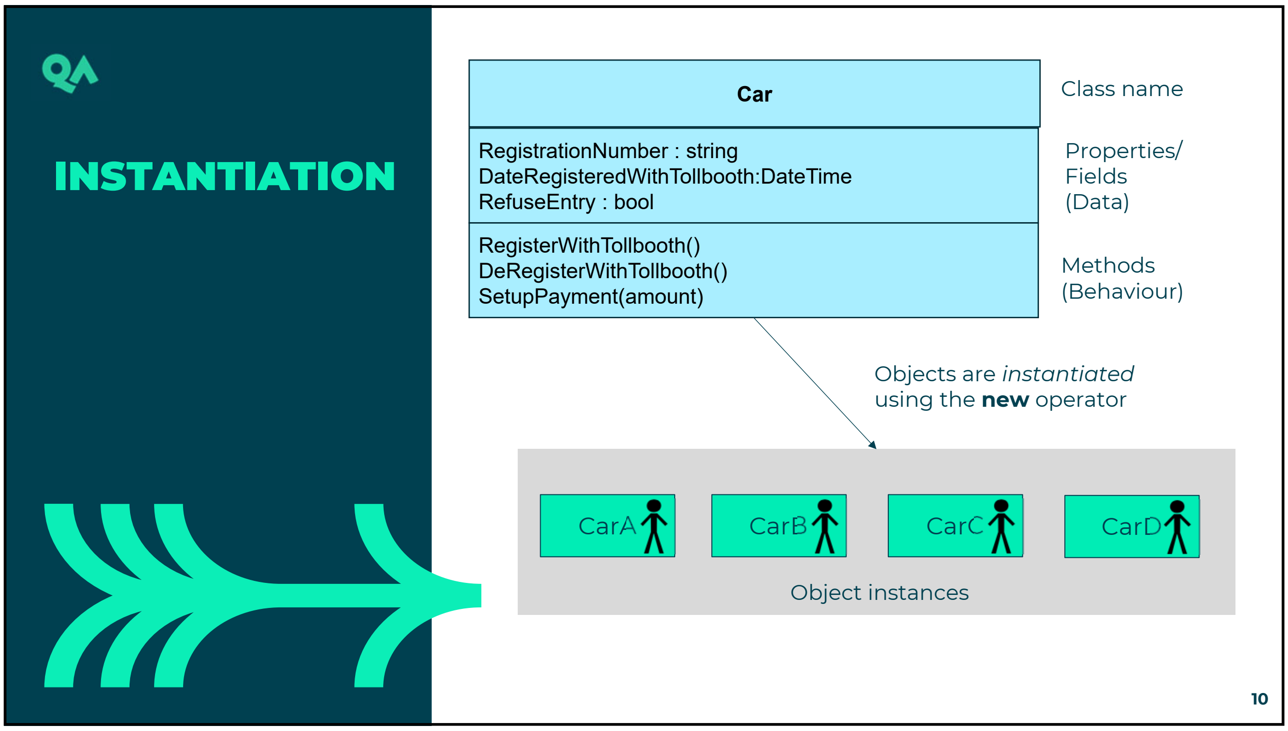
8

8



MULTIPLE INSTANCES

9



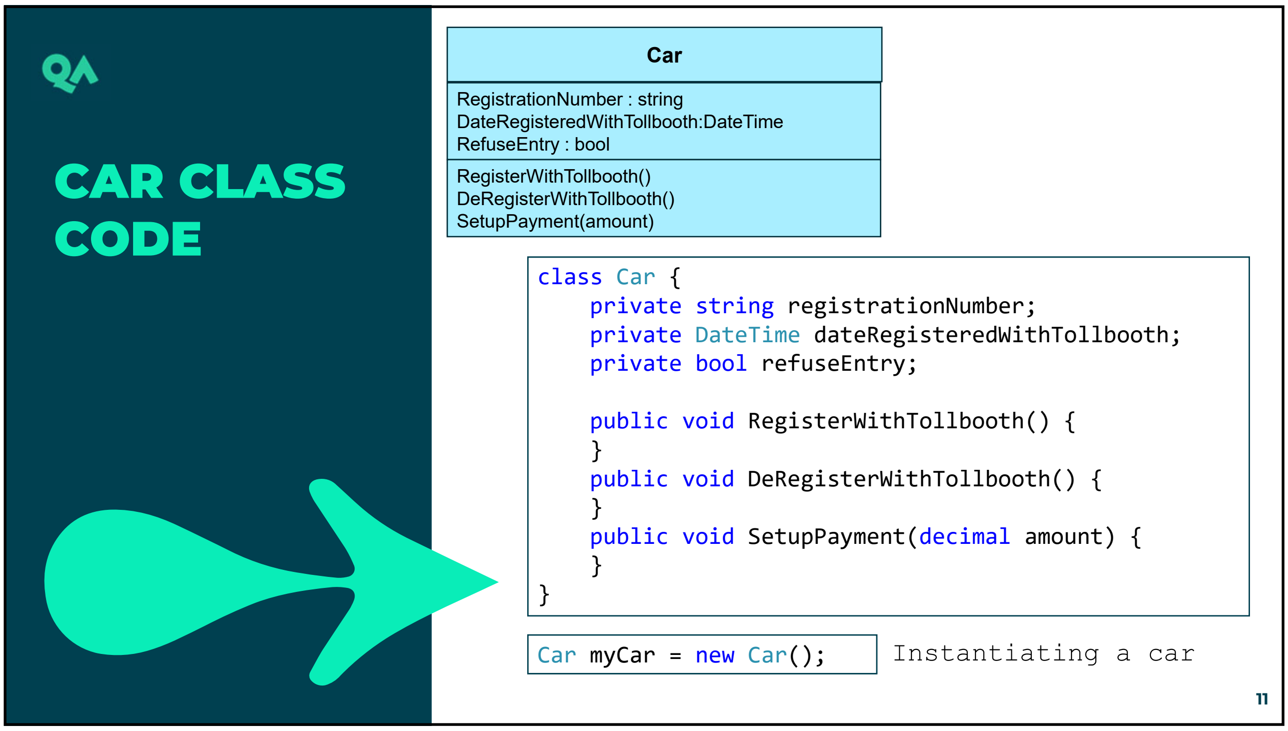
INSTANTIATION

Car
RegistrationNumber : string DateRegisteredWithTollbooth:DateTime RefuseEntry : bool

RegisterWithTollbooth()
DeRegisterWithTollbooth()
SetupPayment(amount)

Class name
Properties/ Fields (Data)
Methods (Behaviour)
Objects are *instantiated* using the **new** operator
CarA
CarB
CarC
CarD
Object instances
10

10



CAR CLASS CODE

Car

RegistrationNumber : string
DateRegisteredWithTollbooth:DateTime
RefuseEntry : bool

RegisterWithTollbooth()
DeRegisterWithTollbooth()
SetupPayment(amount)

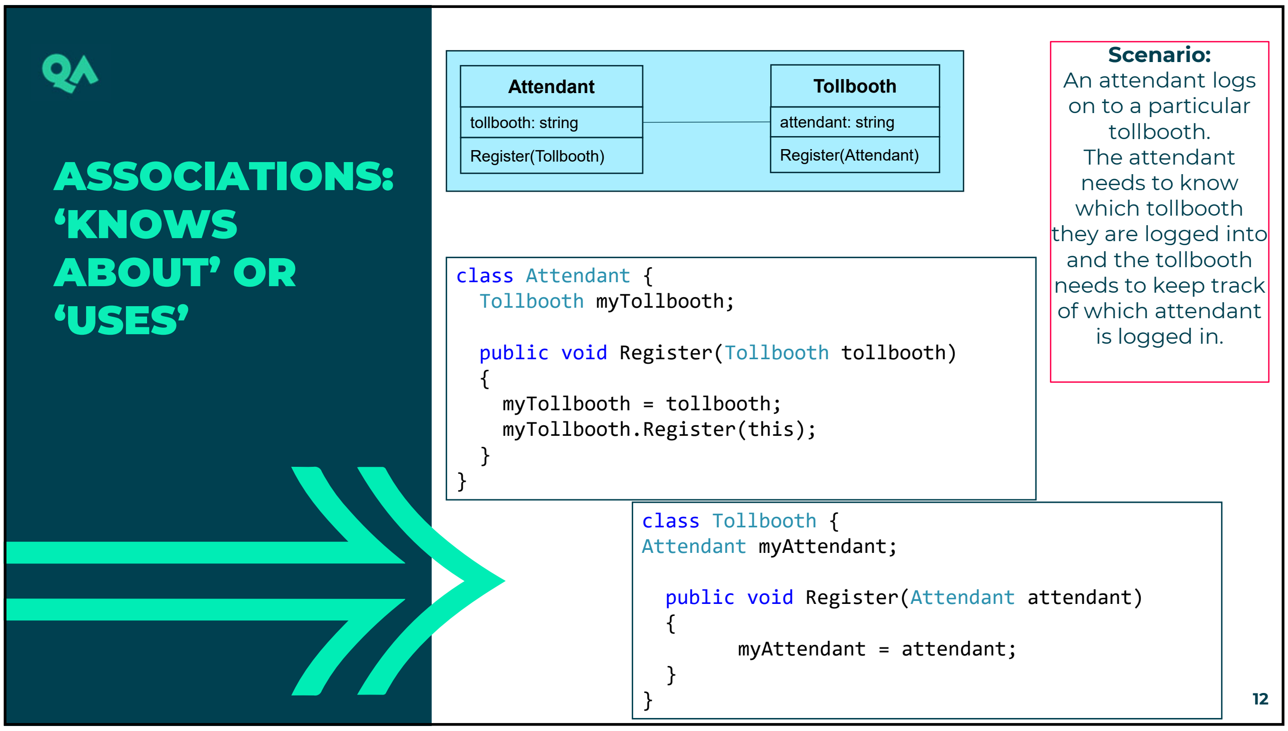
```
class Car {
    private string registrationNumber;
    private DateTime dateRegisteredWithTollbooth;
    private bool refuseEntry;

    public void RegisterWithTollbooth() {
    }
    public void DeRegisterWithTollbooth() {
    }
    public void SetupPayment(decimal amount) {
    }
}
```

```
Car myCar = new Car();
```

Instantiating a car

11

11



ASSOCIATIONS: ‘KNOWS ABOUT’ OR ‘USES’

Attendant

tollbooth: string

Register(Tollbooth)

Tollbooth

attendant: string

Register(Attendant)

Scenario:
An attendant logs on to a particular tollbooth.
The attendant needs to know which tollbooth they are logged into and the tollbooth needs to keep track of which attendant is logged in.

```
class Attendant {
  Tollbooth myTollbooth;

  public void Register(Tollbooth tollbooth)
  {
    myTollbooth = tollbooth;
    myTollbooth.Register(this);
  }
}
```

```
class Tollbooth {
Attendant myAttendant;

  public void Register(Attendant attendant)
  {
        myAttendant = attendant;
  }
}
```

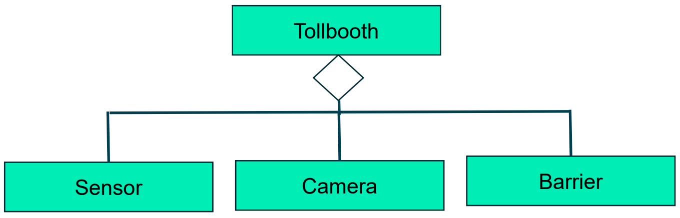
12

12

AGGREGATION: 'HAS A'

Sometimes there is a container object which is composed of other objects. It does its work by asking its subordinate objects to do work.

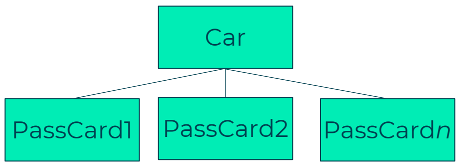
We show this association on a UML class diagram with a diamond (**aggregation** symbol). The diamond is unfilled to denote aggregation and filled to denote **composition**. A tollbooth is composed of a sensor, a camera, and a barrier.



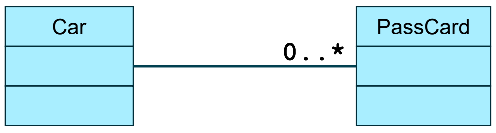
13

MULTIPLICITY

- Multiplicity represents the number of objects that form a relationship within our system



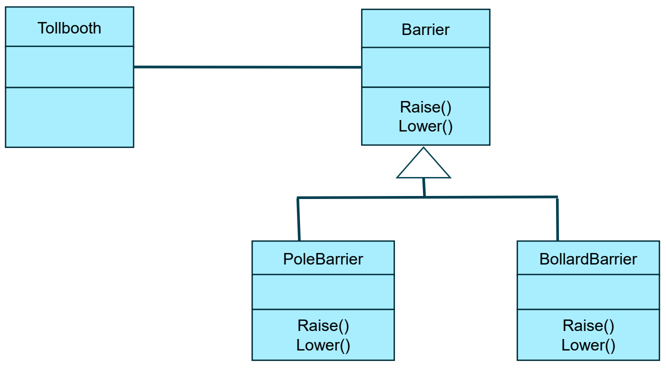
- 'A car can have zero or many pass cards' would be modelled as:



14

INHERITANCE 'IS A' OR 'IS A KIND OF'

- Inheritance is basing an object or class upon another object
- The top-level class is called the ***base*** class or ***superclass***
- The inherited class is called the ***derived*** class or ***subclass***

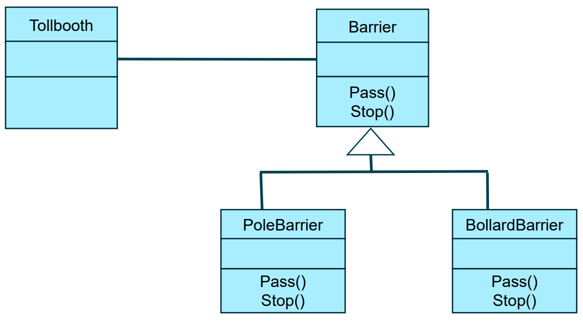


15

15

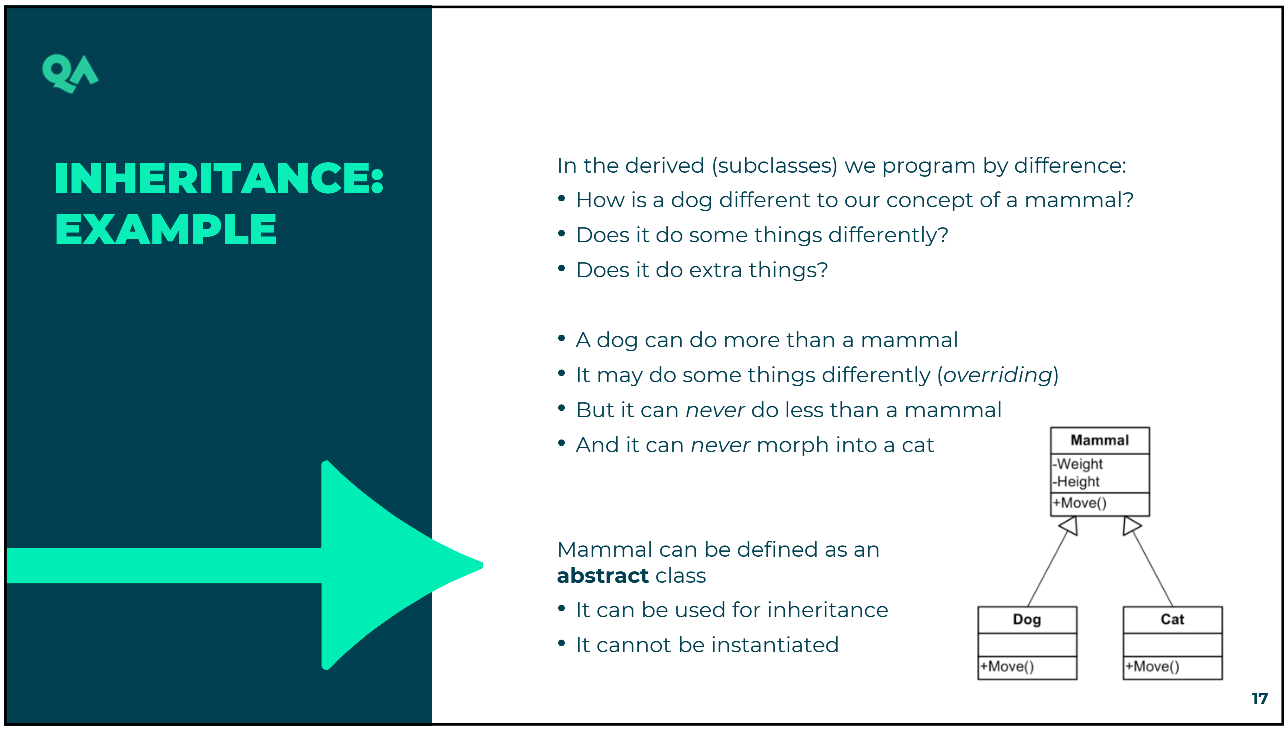
INHERITANCE: IMPLEMENTATION SHOULD BE ENCAPSULATED

- Implementation of a subclass should be well encapsulated and not creep into the superclass
  - A PoleBarier raises to allow a car to *pass*
  - A PoleBarrier lowers to *stop* a car
  - A BollardBarrier raises to *stop* a car
  - A BallardBarrier lowers to allow a car to *pass*



16

16



INHERITANCE: EXAMPLE

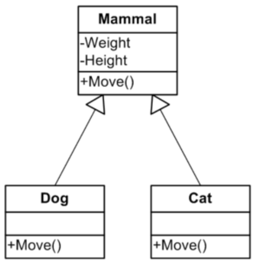
In the derived (subclasses) we program by difference:

- How is a dog different to our concept of a mammal?
- Does it do some things differently?
- Does it do extra things?

- A dog can do more than a mammal
- It may do some things differently (*overriding*)
- But it can *never* do less than a mammal
- And it can *never* morph into a cat

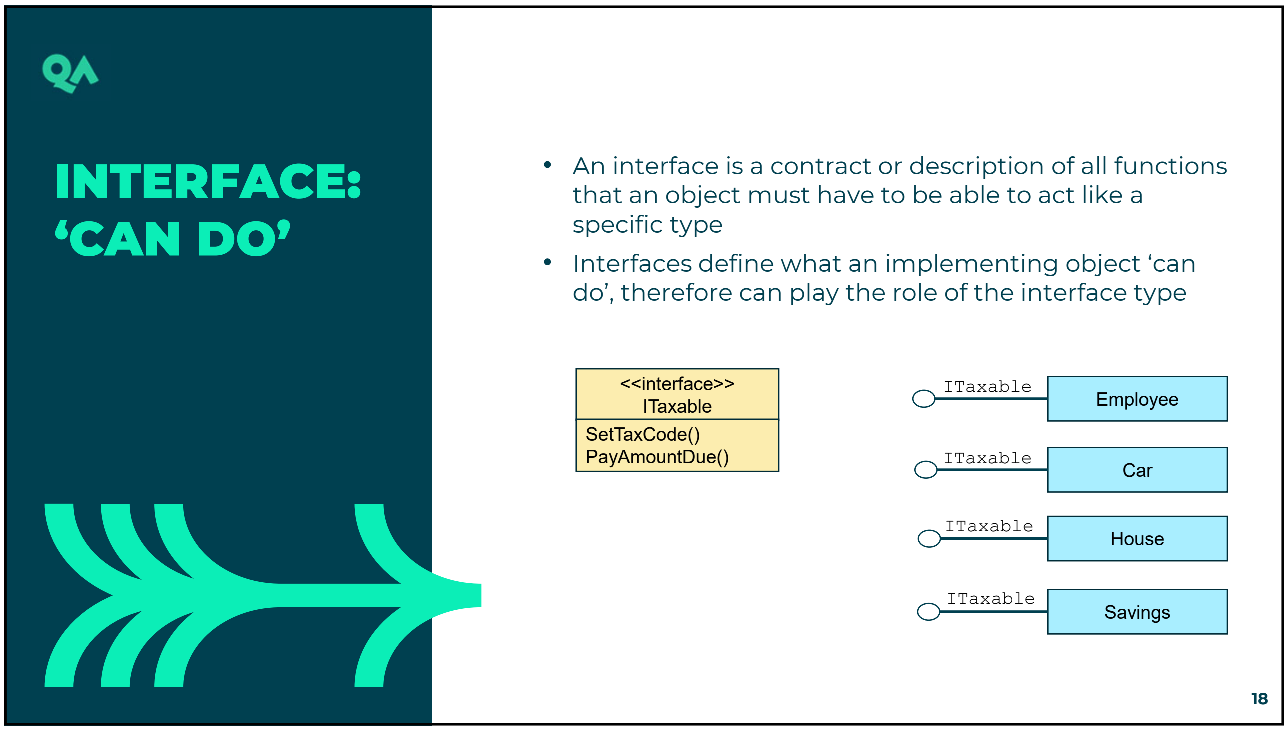
Mammal can be defined as an **abstract** class

- It can be used for inheritance
- It cannot be instantiated



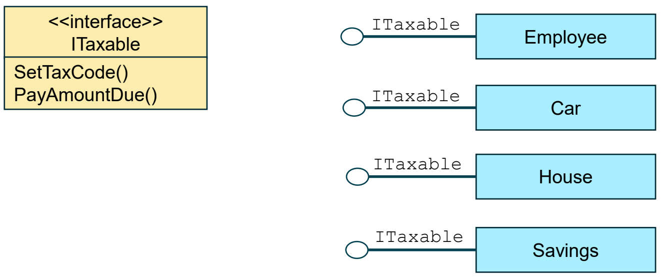
17

17



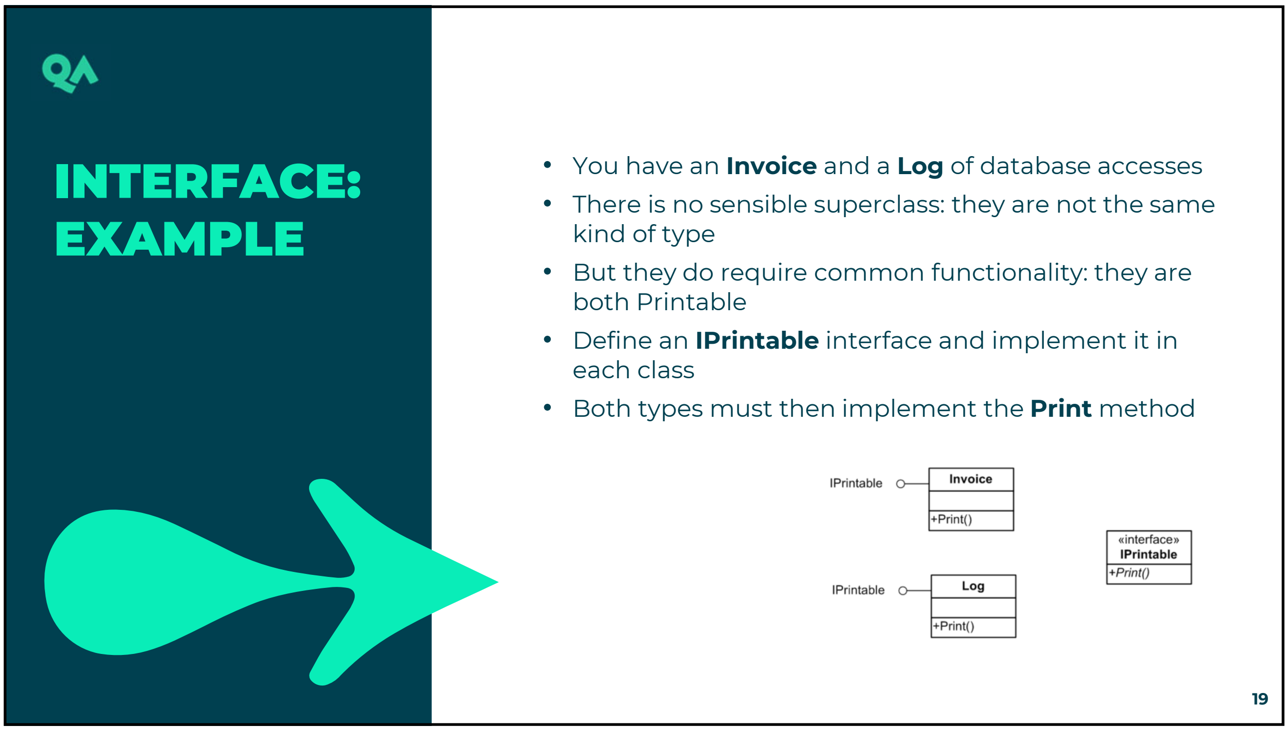
INTERFACE: 'CAN DO'

- An interface is a contract or description of all functions that an object must have to be able to act like a specific type
- Interfaces define what an implementing object 'can do', therefore can play the role of the interface type



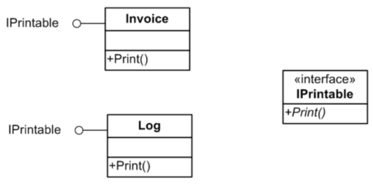
18

18



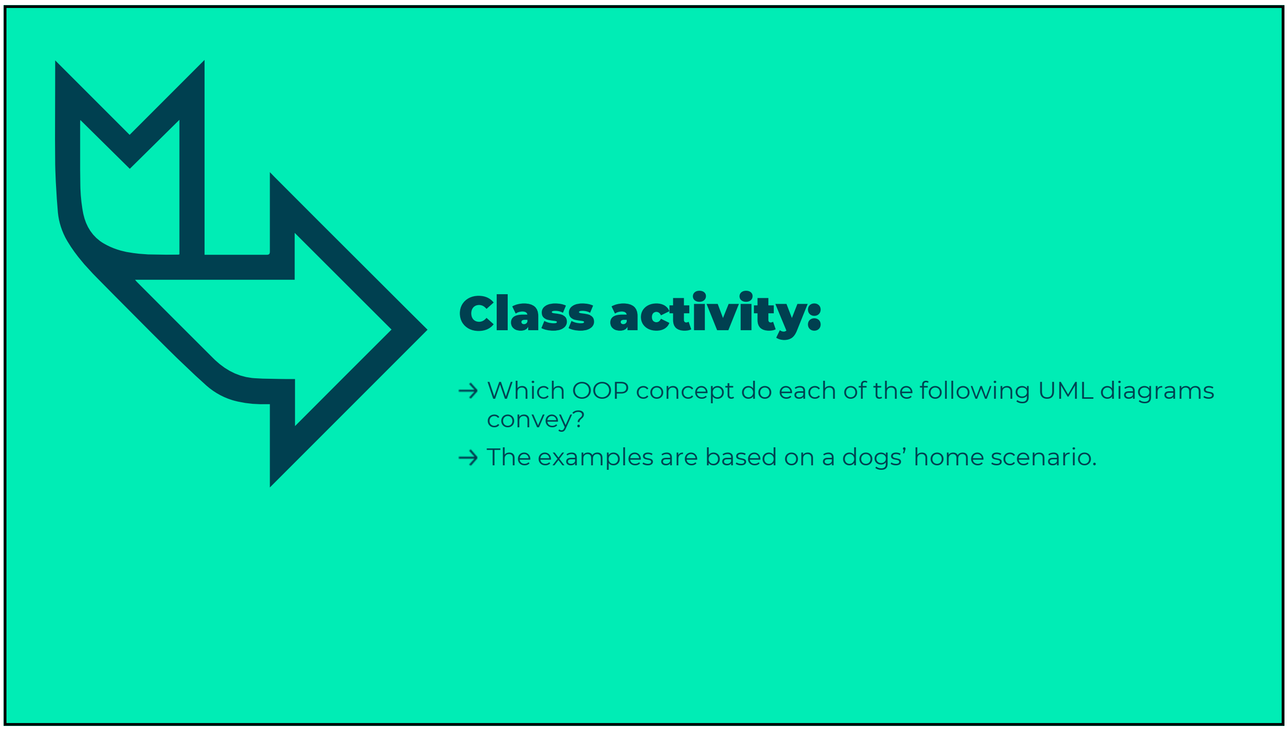
INTERFACE: EXAMPLE

- You have an **Invoice** and a **Log** of database accesses
- There is no sensible superclass: they are not the same kind of type
- But they do require common functionality: they are both Printable
- Define an **IPrintable** interface and implement it in each class
- Both types must then implement the **Print** method



19

19



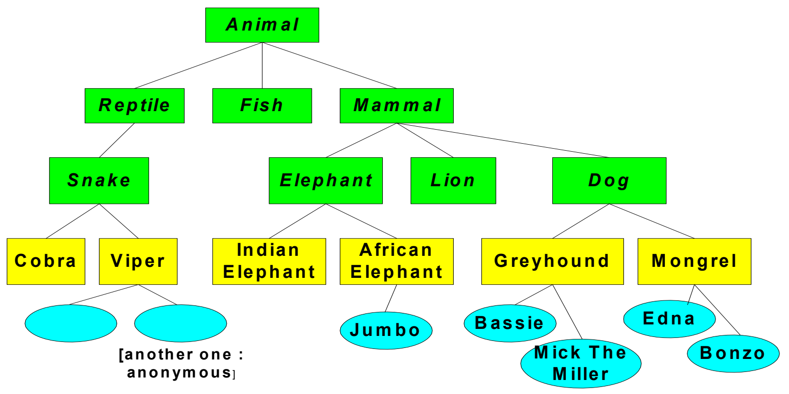
Class activity:

→ Which OOP concept do each of the following UML diagrams convey?

→ The examples are based on a dogs' home scenario.

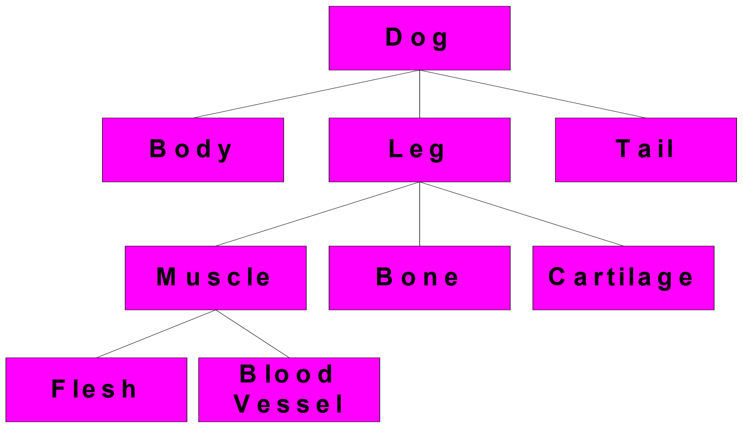
20

CONCEPT 1



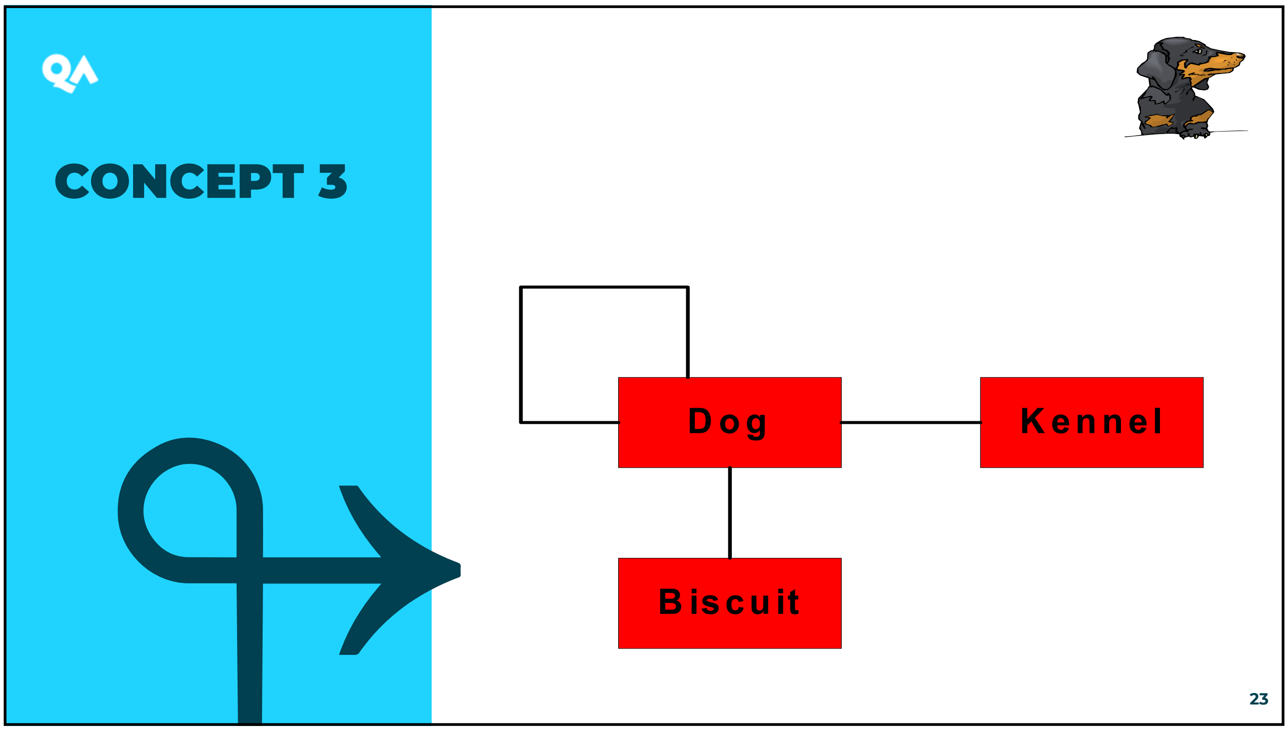
21

CONCEPT 2

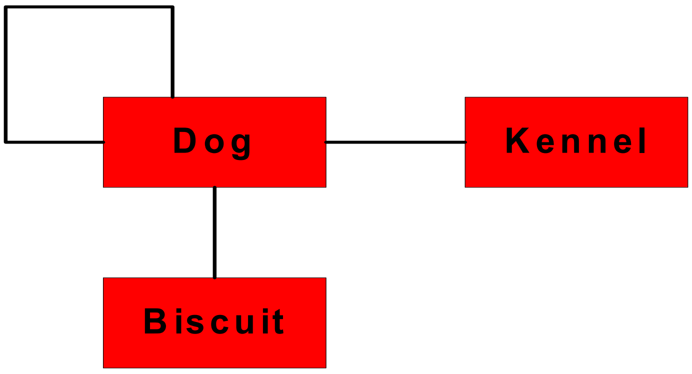


22

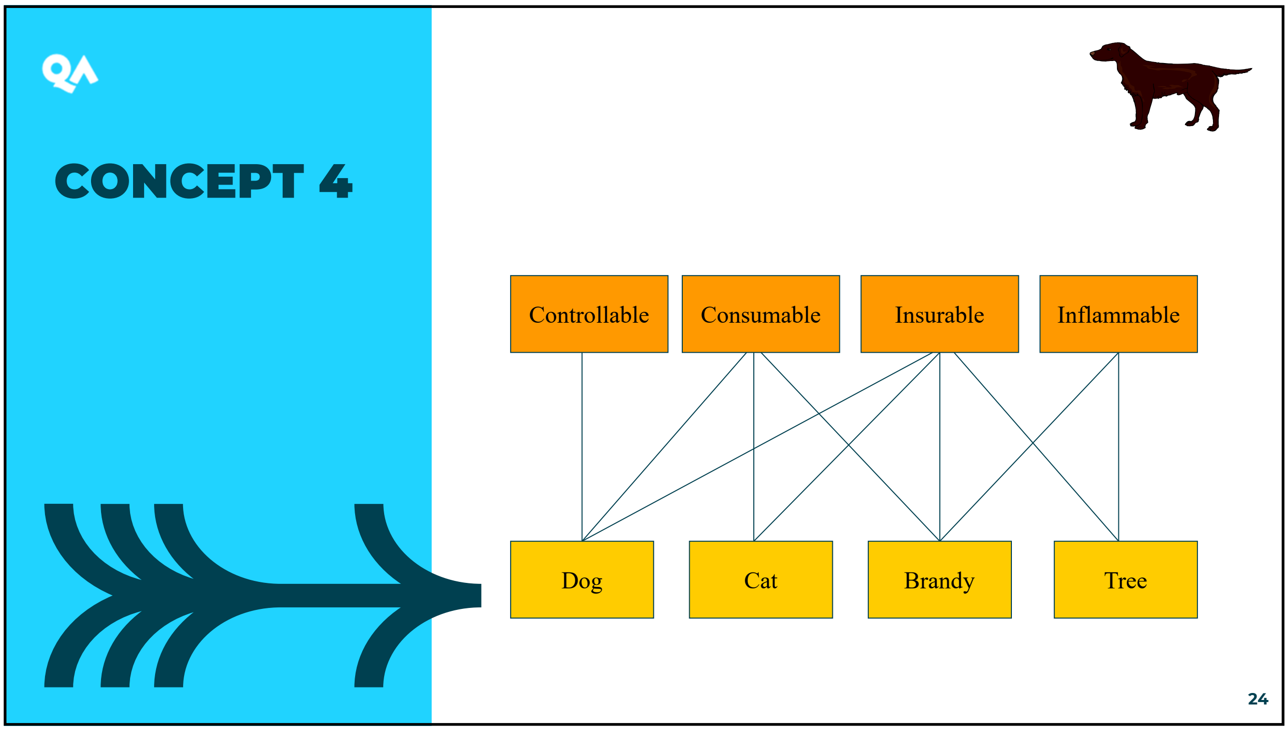
22



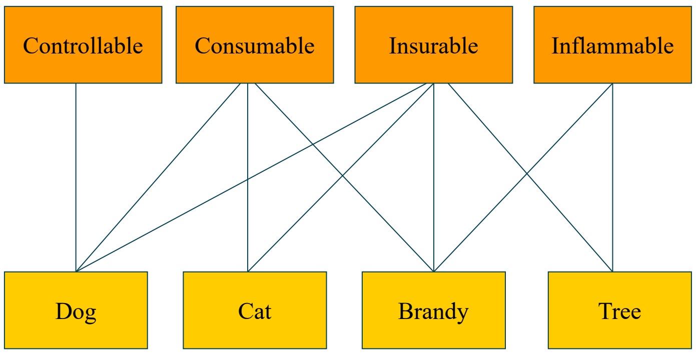
CONCEPT 3



23

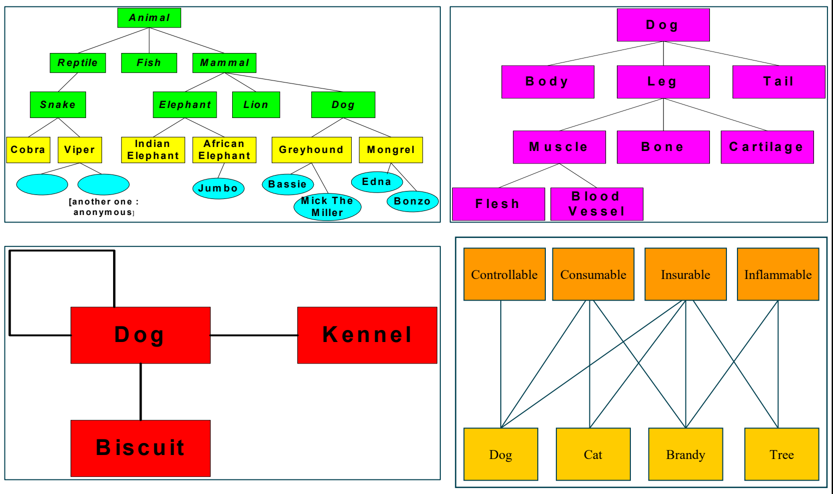


CONCEPT 4



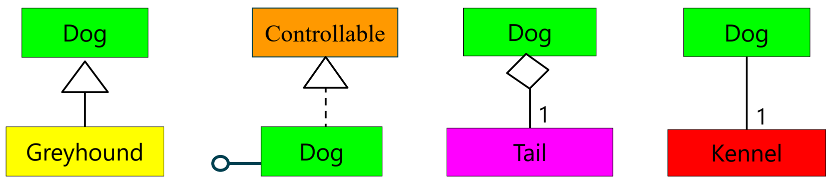
24

ALL CONCEPTS



25

UML NOTATION

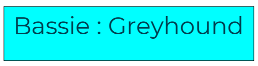


INHERITANCE
Greyhound
'is a kind of'
Dog

INTERFACE
Dog
'exhibits the behaviour of'
Controllable

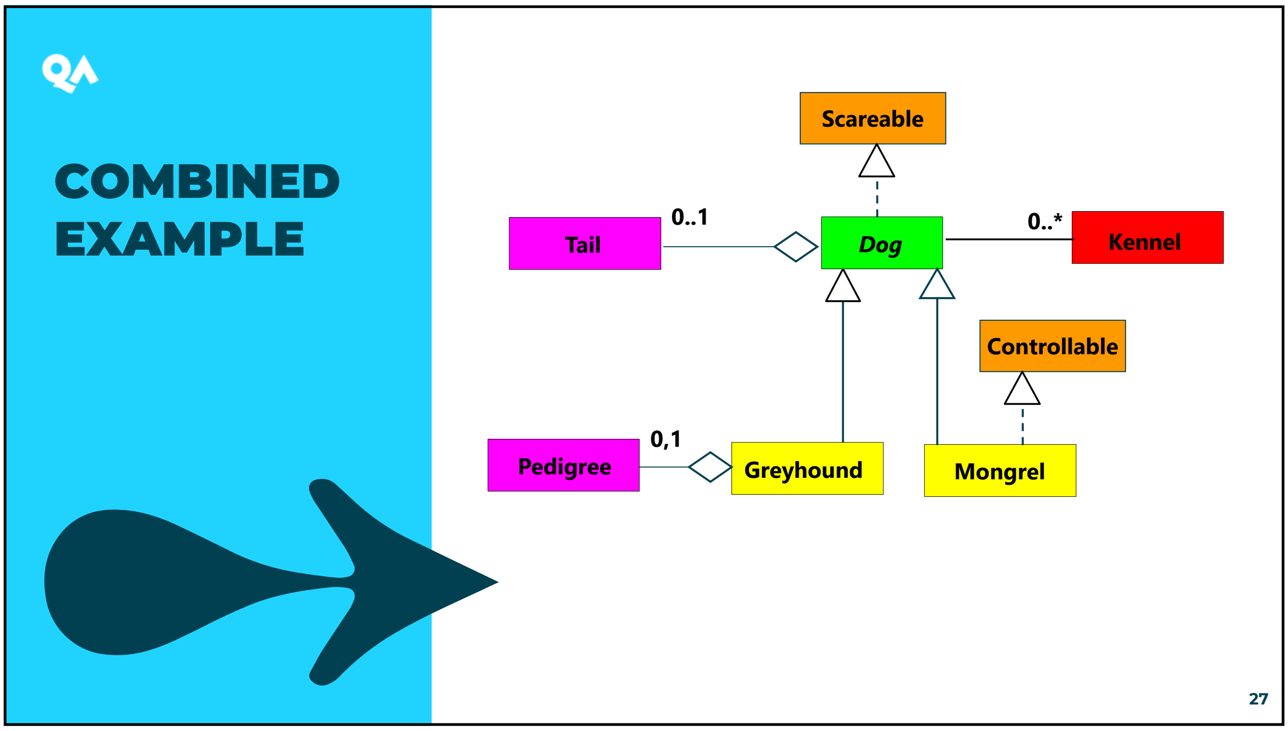
AGGREGATION
Dog
'has the object'
Tail

ASSOCIATION
Dog
'knows about the object'
Kennel



INSTANCE
Bassie
'is an example of a'
Greyhound

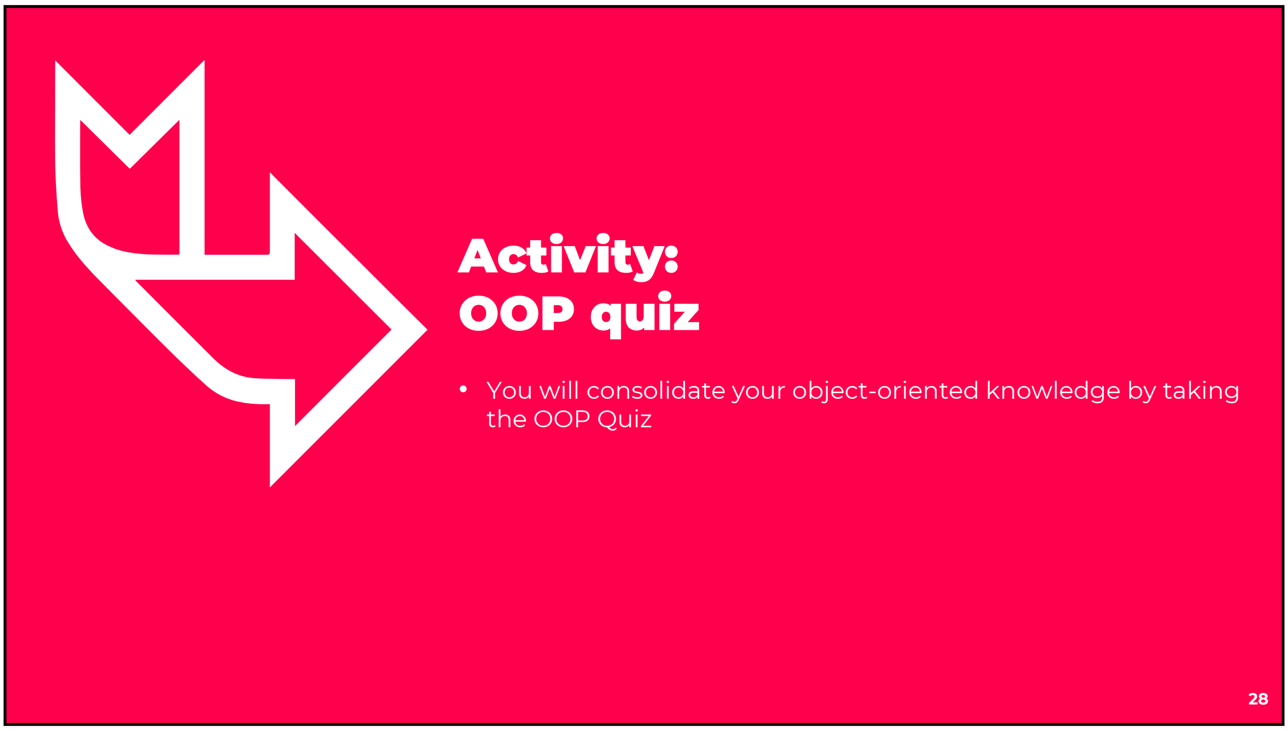
26



COMBINED EXAMPLE

27

27

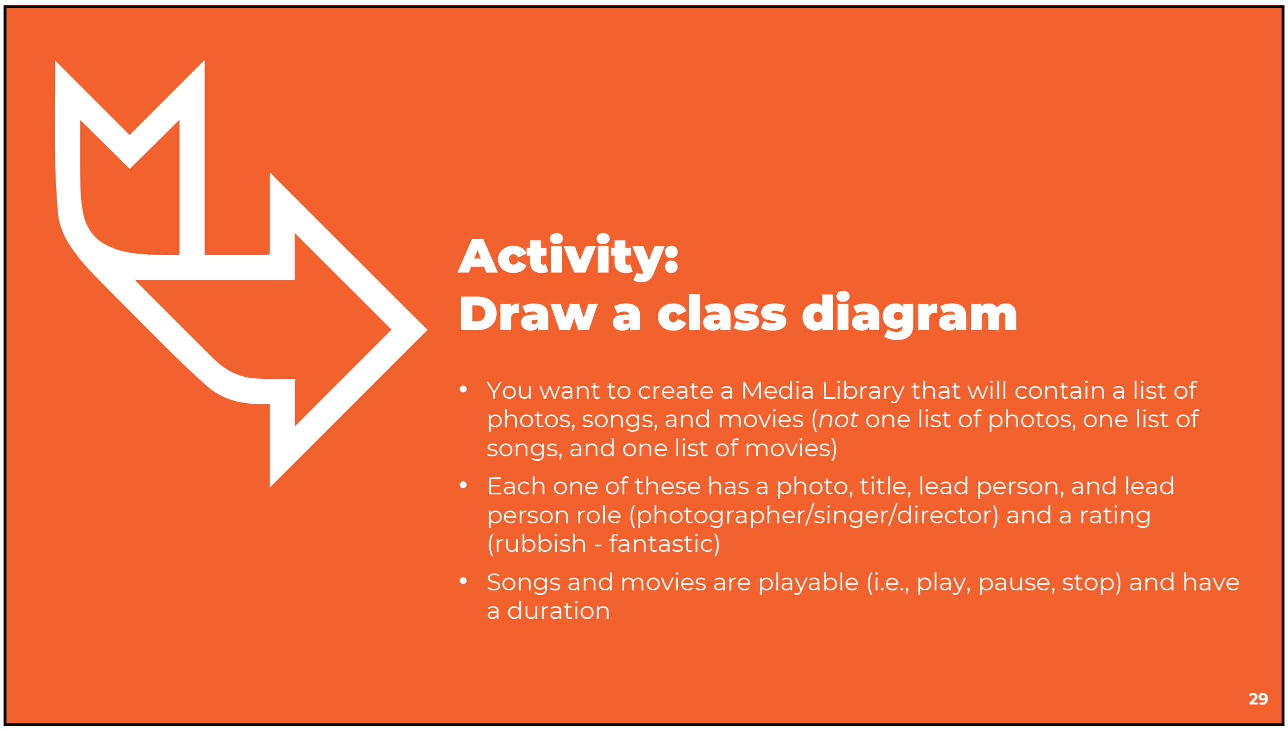


Activity: OOP quiz

- You will consolidate your object-oriented knowledge by taking the OOP Quiz

28

28



Activity:
Draw a class diagram

- You want to create a Media Library that will contain a list of photos, songs, and movies (*not* one list of photos, one list of songs, and one list of movies)
- Each one of these has a photo, title, lead person, and lead person role (photographer/singer/director) and a rating (rubbish - fantastic)
- Songs and movies are playable (i.e., play, pause, stop) and have a duration

29

29

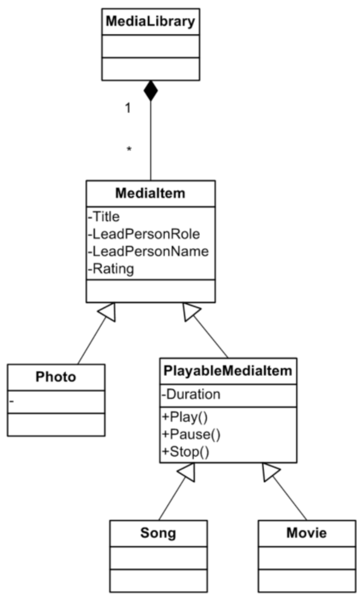
Activity:
Solution

- Two possible solutions follow

30

30

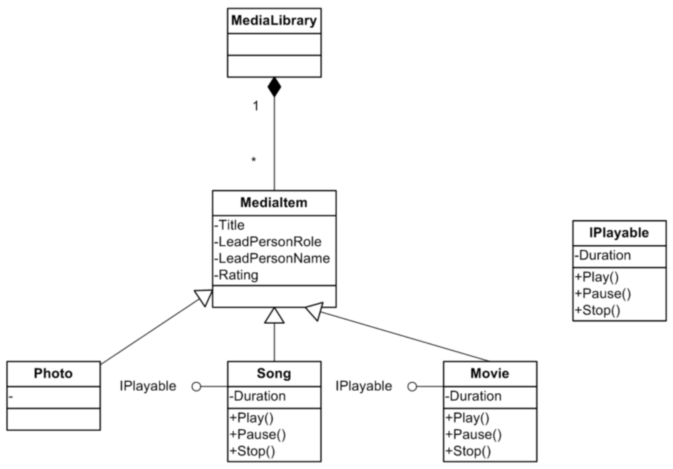
SOLUTION 1



31

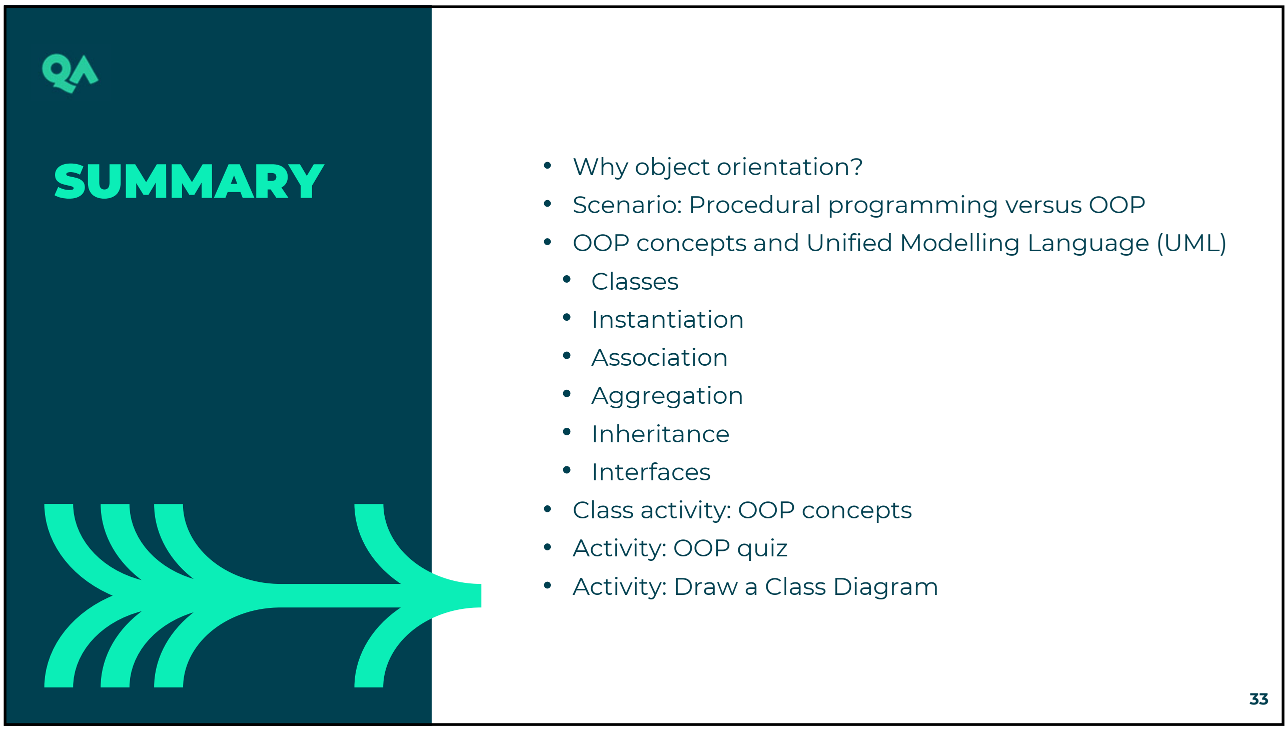
31

SOLUTION 2



32

32



SUMMARY

- Why object orientation?
- Scenario: Procedural programming versus OOP
- OOP concepts and Unified Modelling Language (UML)
  - Classes
  - Instantiation
  - Association
  - Aggregation
  - Inheritance
  - Interfaces
- Class activity: OOP concepts
- Activity: OOP quiz
- Activity: Draw a Class Diagram

33

33