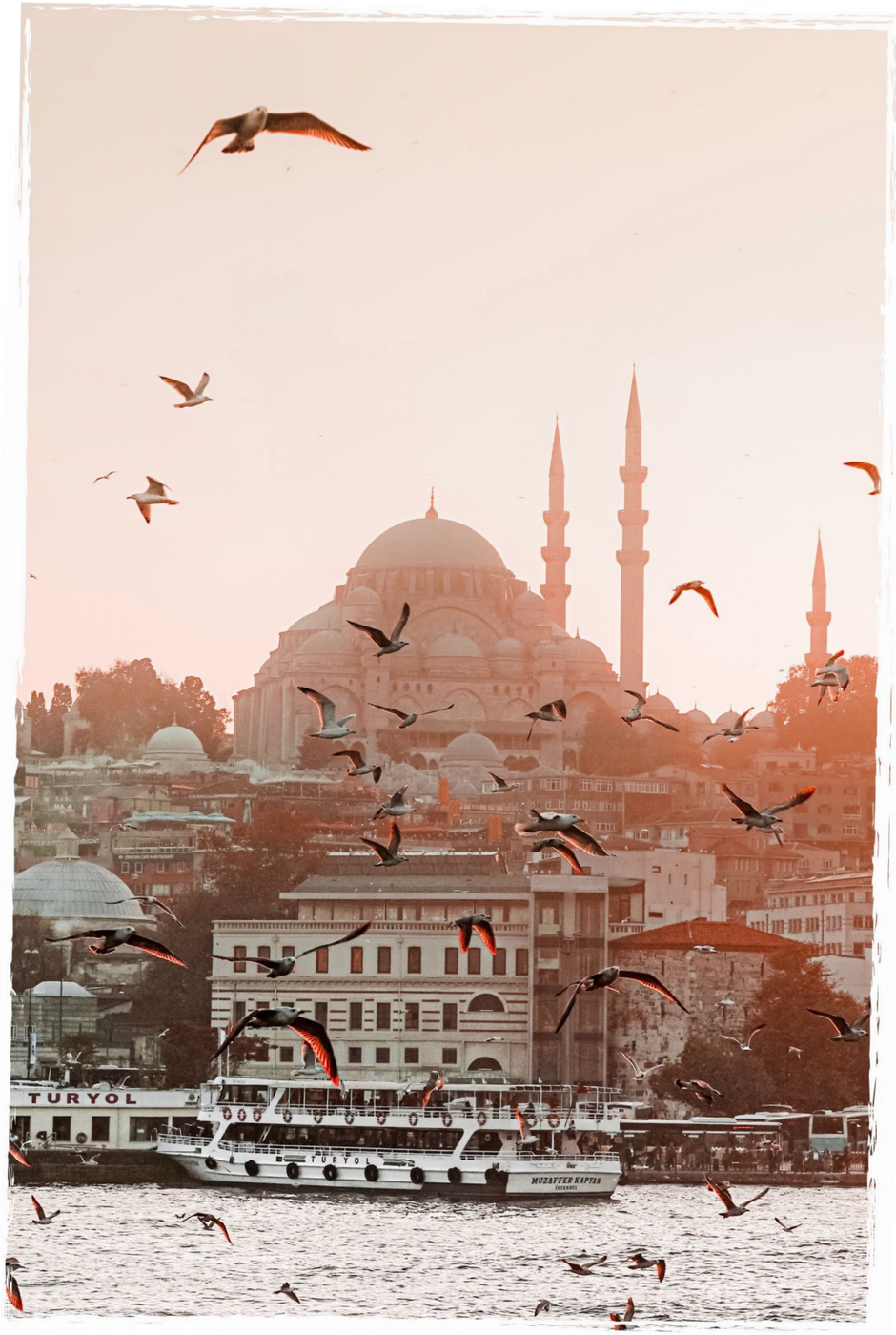
COMP1007

Introduction to Object Orientation

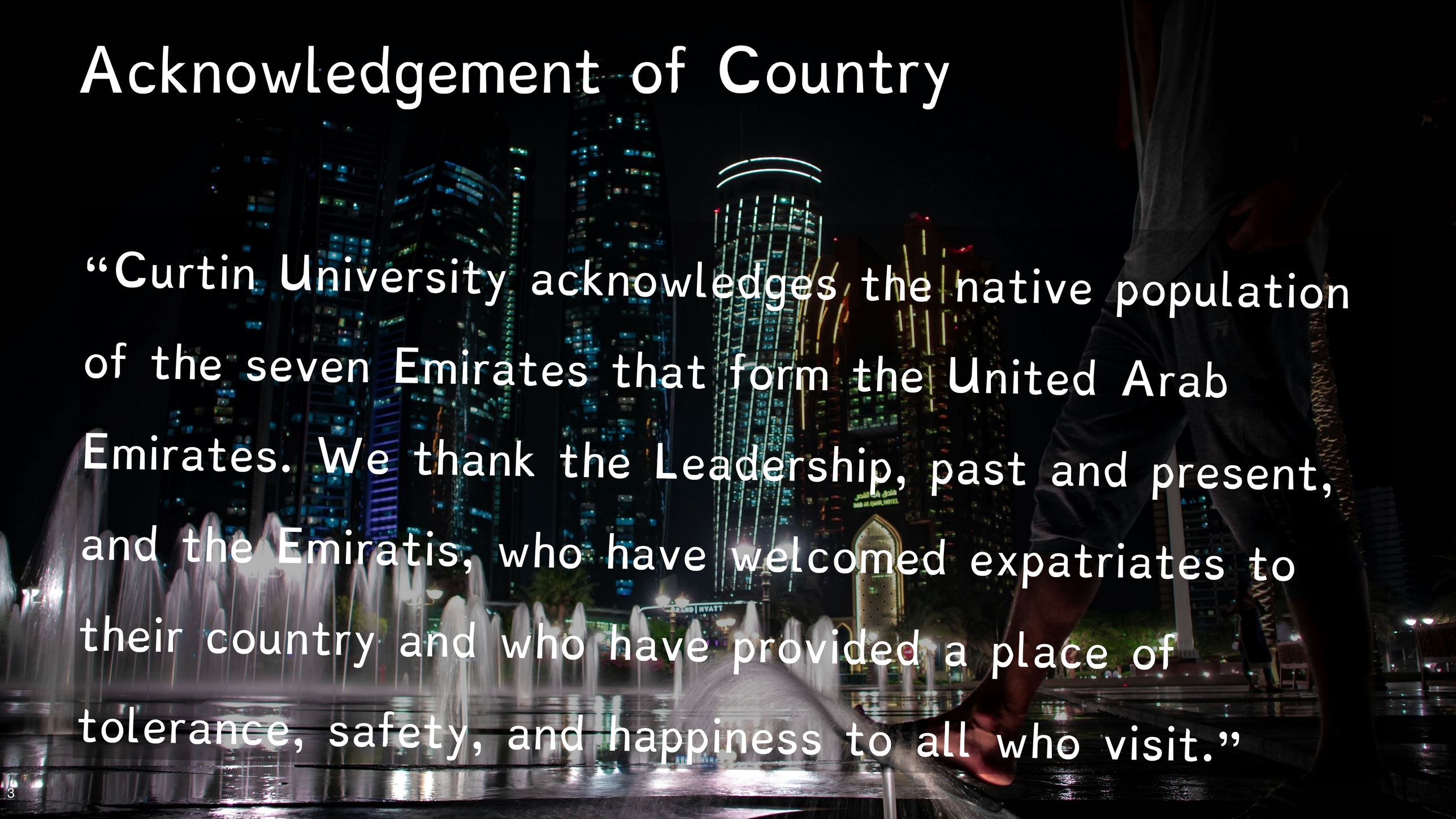
Dr David A. McMeekin



Acknowledgement of Country

“I acknowledge the Wadjuk people of the Noongar nation on which Curtin University’s Boorloo Campus sits, and the Wangkatja people where the Karlkurla Campus sits, as the traditional custodians of the lands and waterways, and pay my respect to elders past, present and emerging.”

Acknowledgement of Country



“Curtin University acknowledges the native population of the seven Emirates that form the United Arab Emirates. We thank the Leadership, past and present, and the Emiratis, who have welcomed expatriates to their country and who have provided a place of tolerance, safety, and happiness to all who visit.”

COMP1007 - Unit Learning Outcomes

- Identify appropriate primitive data types required for the translation of pseudocode algorithms into Java;
- Design in pseudocode simple classes and implement them in Java in aLinux command-line environment;
- Design in pseudocode and implement in Java structured procedural algorithms in a Linux command-line environment;
- Apply design and programming skills to implement known algorithms in real world applications; and
- Reflect on design choices and communicate design and design decisions in a manner appropriate to the audience.

Outline

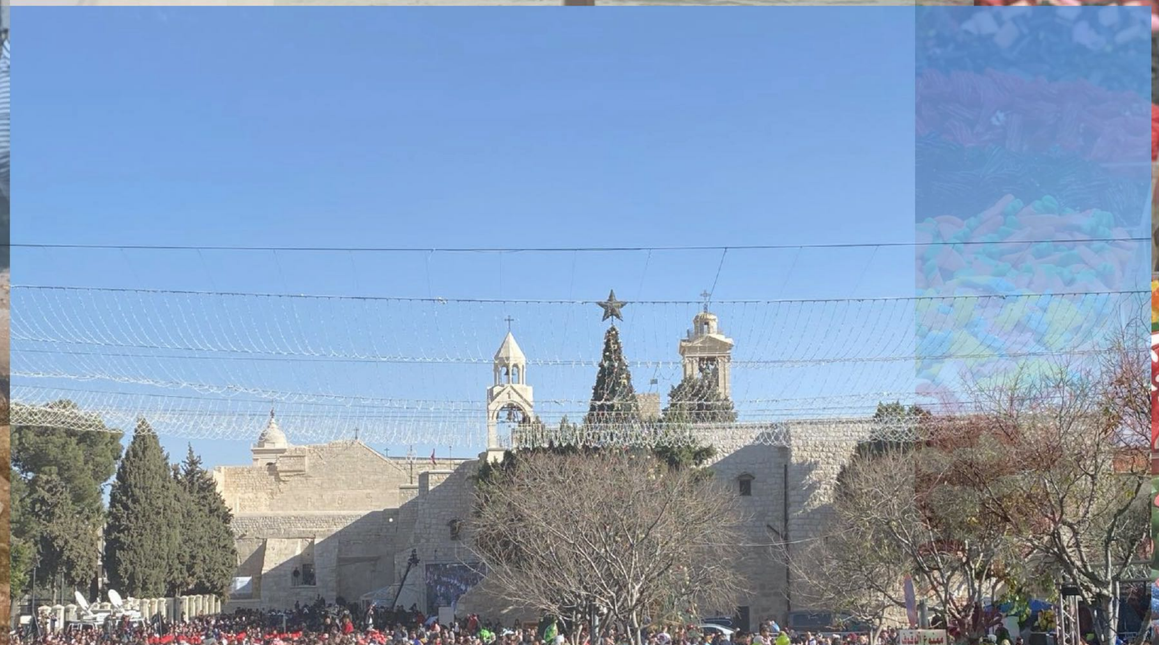
- Humanity
- OOP Basics
- OOP Pillars
- Basics of Classes
- UML
- Pseudocode
- Inheritance
- Setters and Getters

Objects in the Real World

How We See the Real World

- We see the world as objects;
- Our world is made up of objects;
- Look around, objects are everywhere.

What is in the pictures?



The Objects

- Pens;
- Bread;
- Books;
- Sign;
- Speakers;
- Lollies;
- People;
- ...

The Objects' Details

- Pens: colour, ink, full, empty...
- Bread: brown, white, loaf, rolls...
- Books: title, ISBN, publication year...
- Sign: colour, language, languages, text...
- Speakers: colour, height, location...
- Lollies: name, colour, sweet, sour...
- People:...
- There are many attributes to each object.

Objects have Attributes

- Each object has attributes;
- A pen has a colour;
- The colour has a value: purple.
- A book has a title;
- The title has a value: “Java Programming”.

Attributes, Values and State

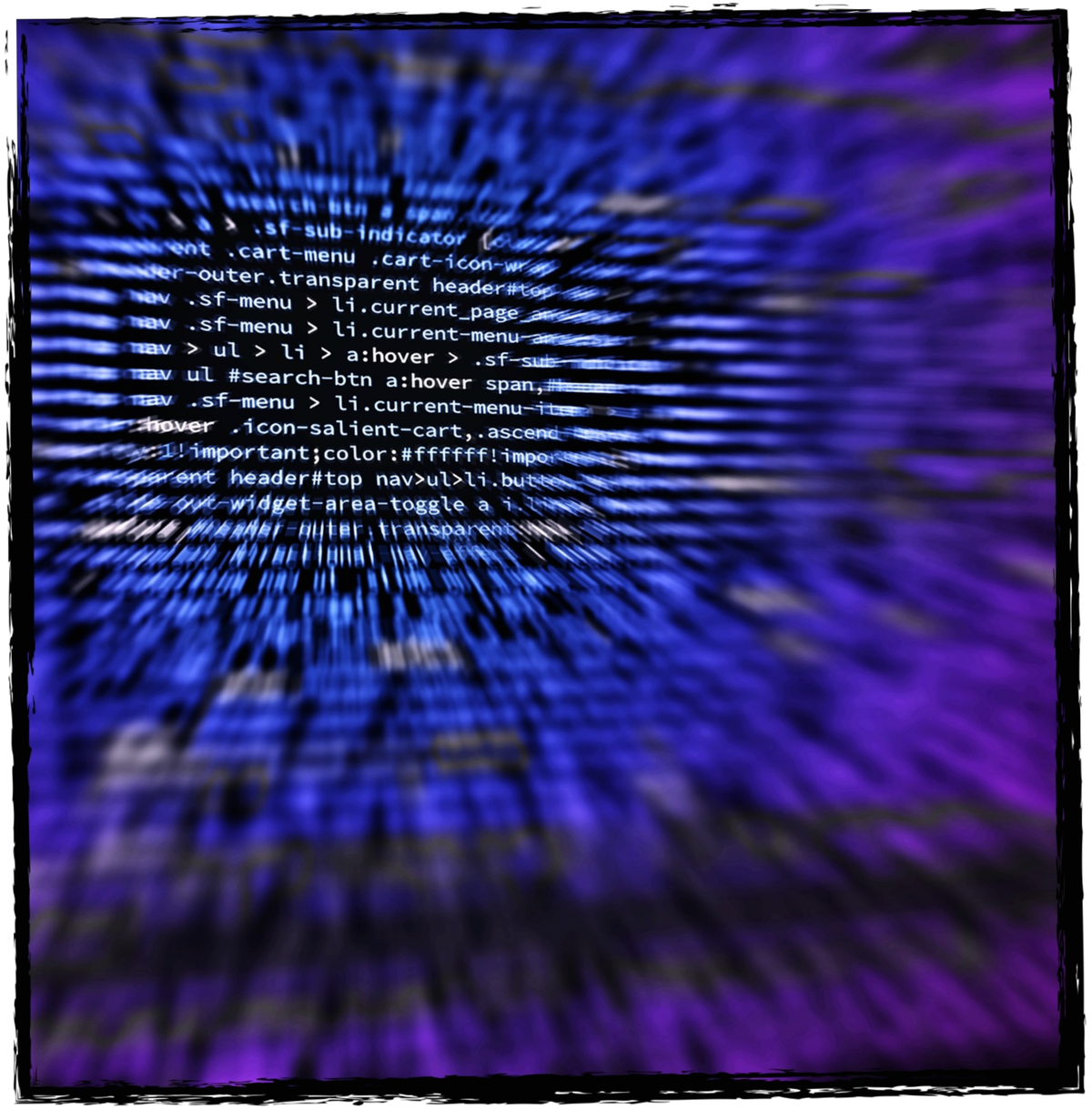
- Attributes: '...the characteristics that define an object...' (Farrell, 2018).
- Values: generally the data in the Attributes;
- State: the set of all the values in the Attributes.

Example: the State of the Game

- A football match occurs in a stadium;
- The football stadium has seats for spectators;
- The state of the stadium: at capacity, match playing;
- A football match has players;
- A football match has a score;
- The value of the score: 3–1;
- The state: Liverpool are winning, Chelsea are not winning.

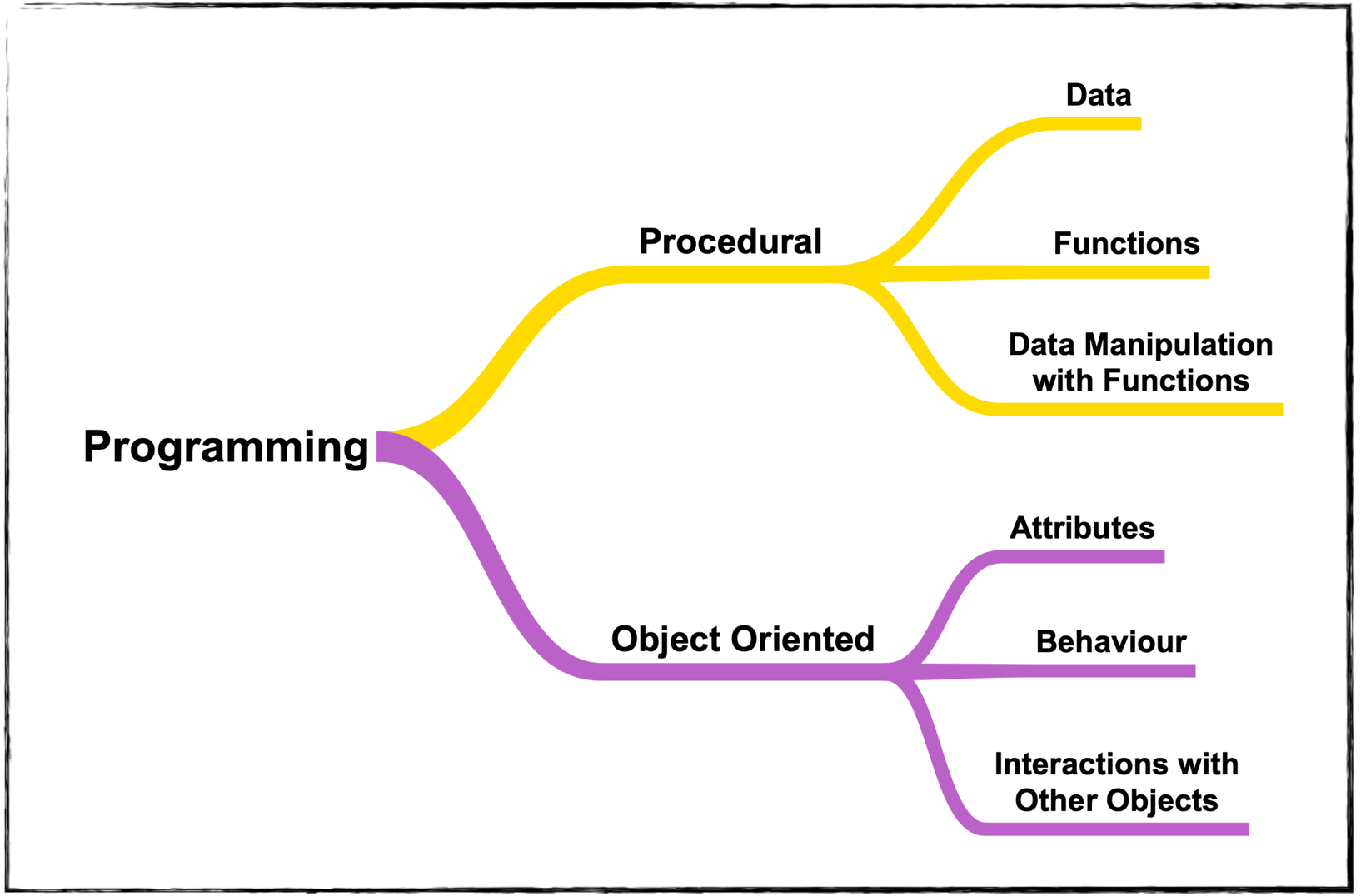
State of the Stadiums

Object Oriented Programming



Programming Paradigms

- Two programming Paradigms discussed in COMP1007:



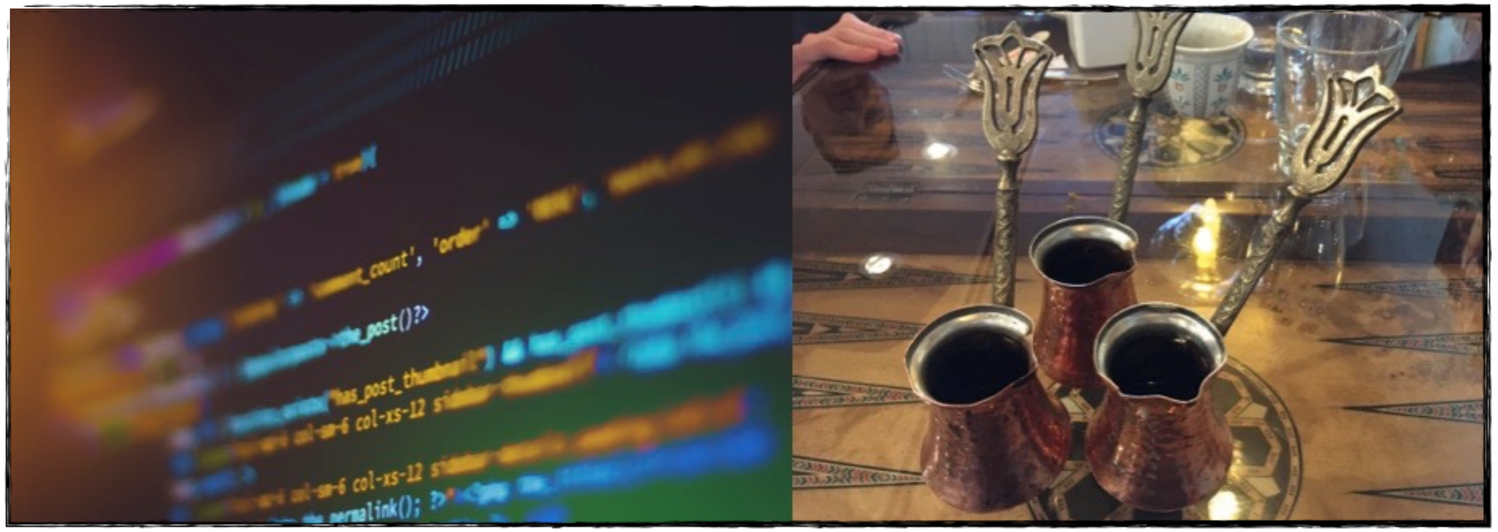
Until Now – Procedural Programming

- All about logic and functions;
- There is data, individual variables, arrays...;
- The variables and arrays have values in them;
- There are algorithms, to do things;
- Algorithms are refined, methods created;
- Steps are repeated;
- The algorithms manipulate the data;
- Results appear from the manipulation;
- The data and algorithms are kept separate from each other.

Object Orientated Programming (OOP)

- Programming paradigm representing real world objects;
- Programming paradigm that attempts to model software from real world objects;
- Focused on an applications's data and methods to manipulate that data;
- The objects to be manipulated are considered.

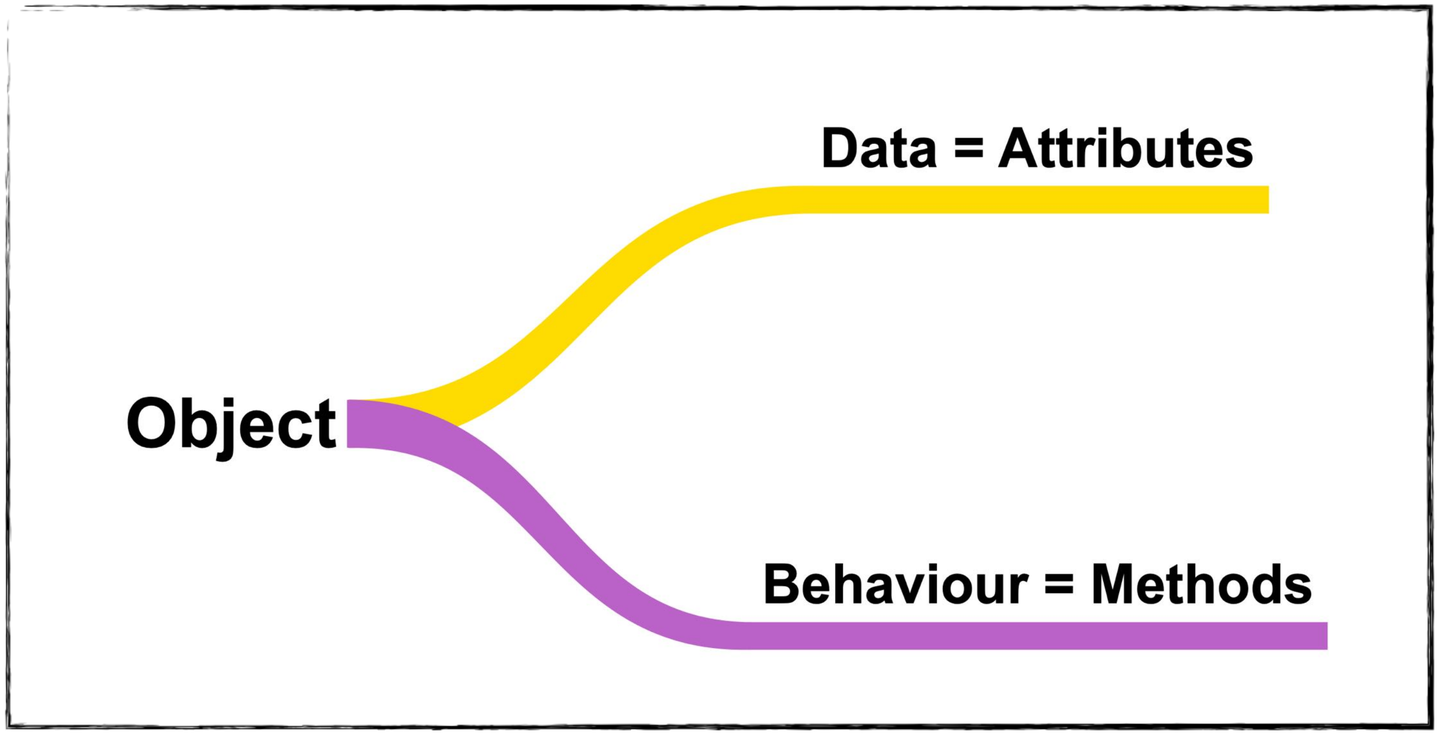
Object Orientated Programming (OOP)



- These things called objects are created;
- Objects contain data;
- Objects contain methods to manipulate their data;
- The data and methods are encapsulated within the object.;
- Objects interact with other objects.

The Basics of Object Oriented Programming

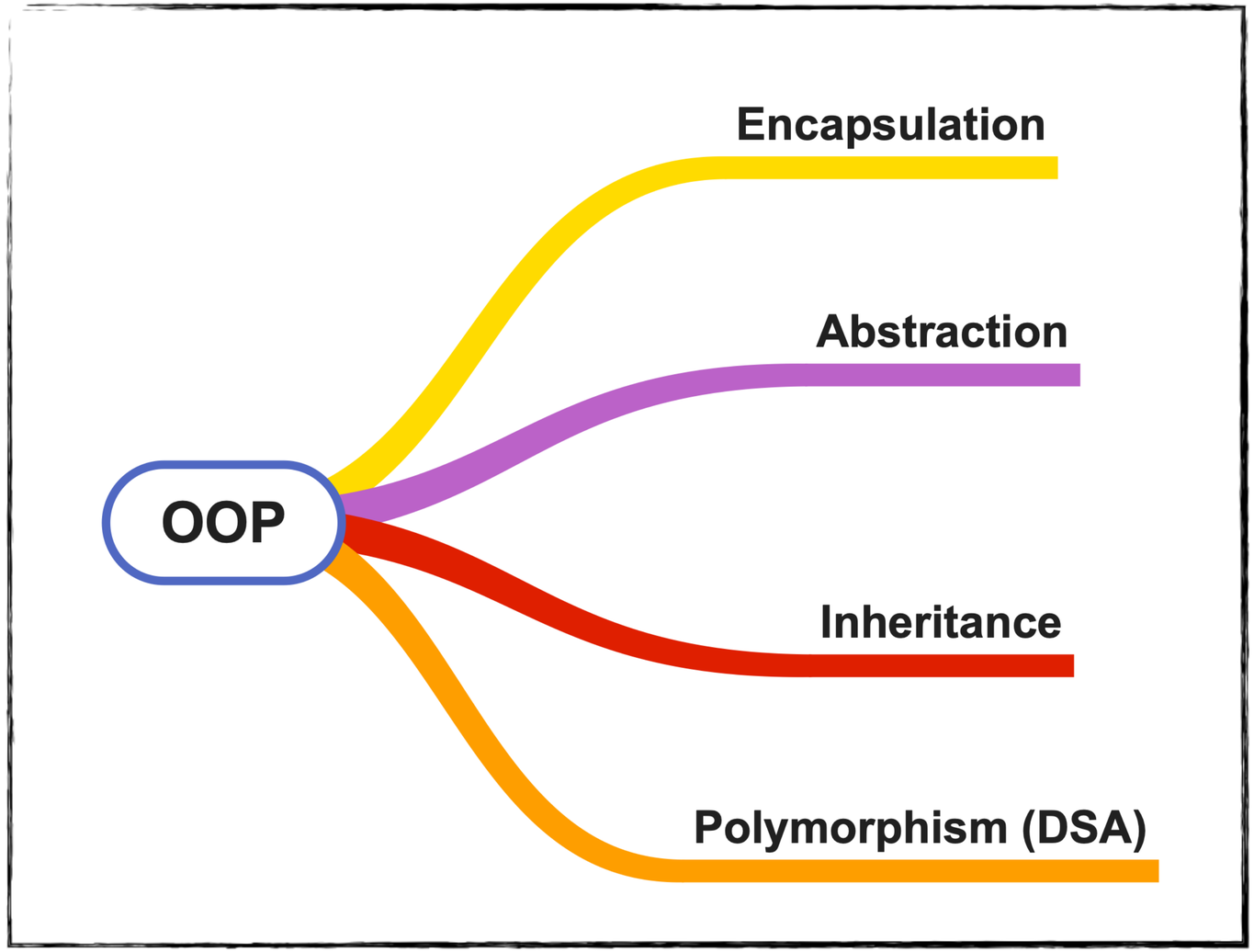
- OOP: about what to manipulate, not how to manipulate it;
- Object Oriented Programming Simplified:



Pillars of OOP

Four Pillars of OOP

- The OOP building blocks:



Encapsulation

- Used to “hide” information from the outside world;
- Is the enclosure of data and methods within an object;
- Allows an object’s methods and data to be treated as a single entity;
- Is concealing an object’s methods and data from external sources;
- Is also know as information hiding.

Abstraction

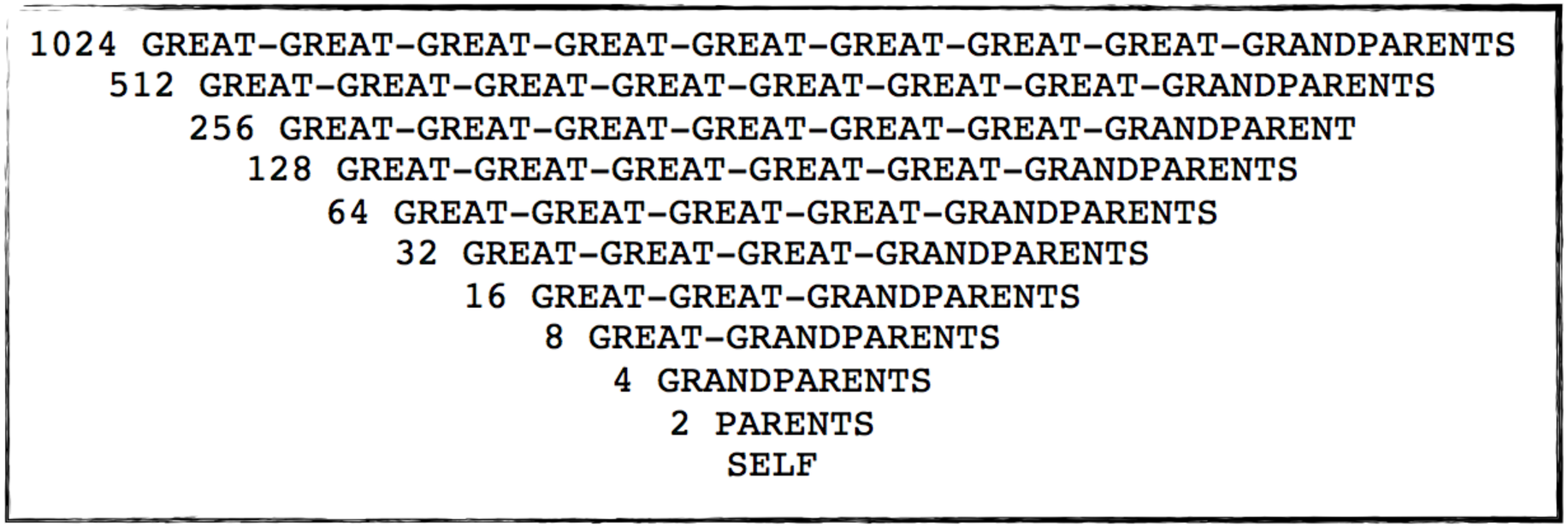
“The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise”

Edsger Dijkstra

Abstraction

- An attempt to reduce complexity;
- Attempts to make complex tasks look simple;
- Attempts to isolate the impact of change.

Inheritance



- Acquiring the traits of predecessors;
- Cars have similar traits to each other;
- Create a parent object, its descendants have all of its traits plus its own unique traits.

Polymorphism

- Open a door;
- Open a book;
- Open a bank account;
- Open a shop;
- Open a ...;
- Same task, done differently.

Classes

Blueprints/Plans

Created from the Blueprint/Plan

- Built from the plan on the previous slide:

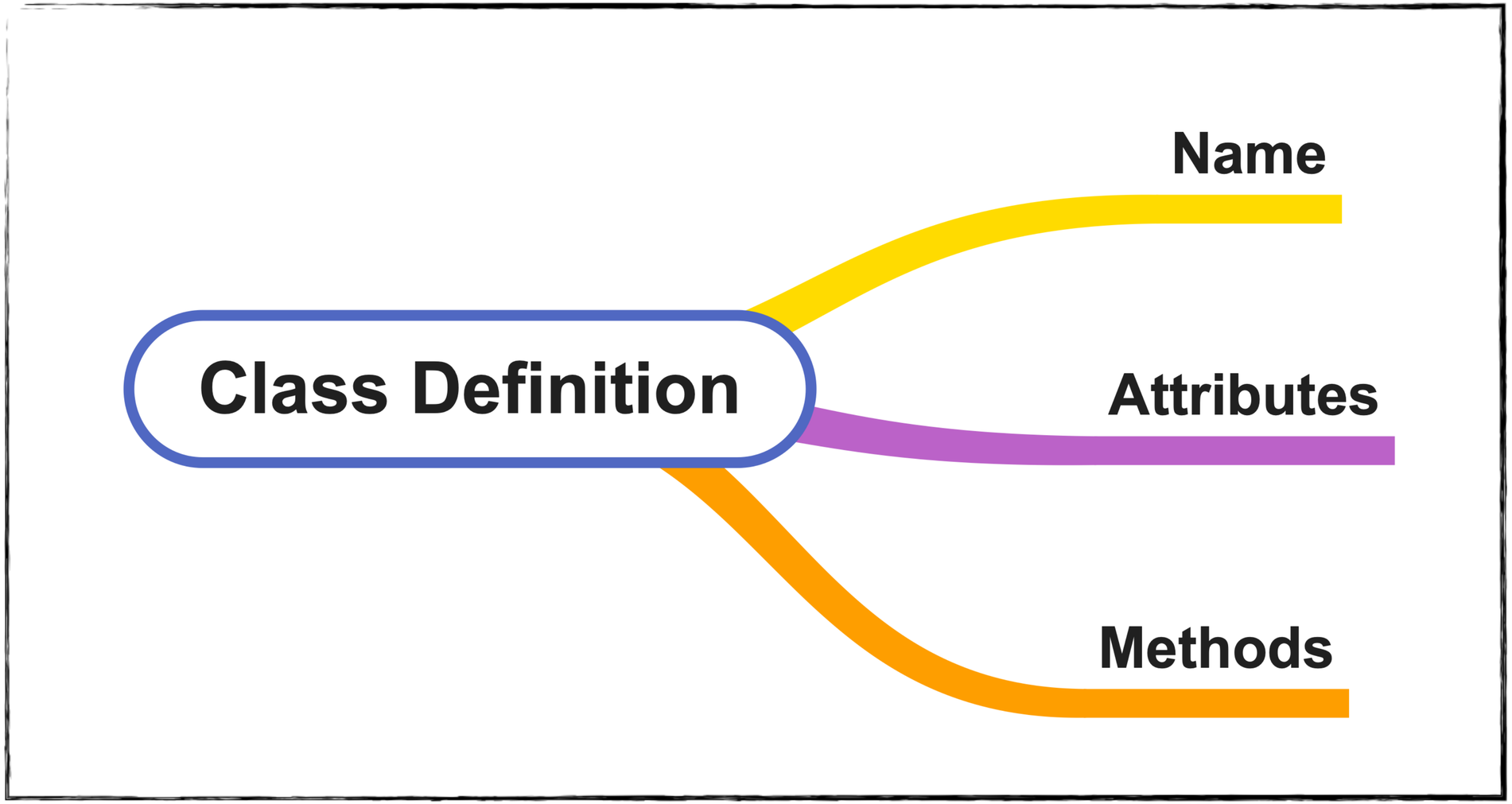
Creating Objects

- Creating something requires a plan or blueprint;
- In OOP, the plan/blueprint is called a class;
- To create an object a class is first needed;
- An object is created from a class.

Nouns and Verbs

- Determining the classes for a system combines art and science;
- One technique: Noun and Verb approach;
- Map nouns to classes;
- Class names describe a thing (e.g., Person).
- Map verbs to methods within classes;
- Method names describe an action (i.e., `getName()`);

Defining Classes



Defining Classes

- Class Definition is generally made up of 3 parts:
  1. Name;
  2. Data; and
  3. Methods.

- Class Definition: a set of program statements listing each objects characteristics and the methods it can use (Farrell, 2013).

Class

- A class specifies the state and behaviour that an Object can have:
  - State: values of the objects attributes:
    - class fields; or
    - instance variables.
  - Behaviour: what the Object can do:
    - Methods; or
    - Functions.

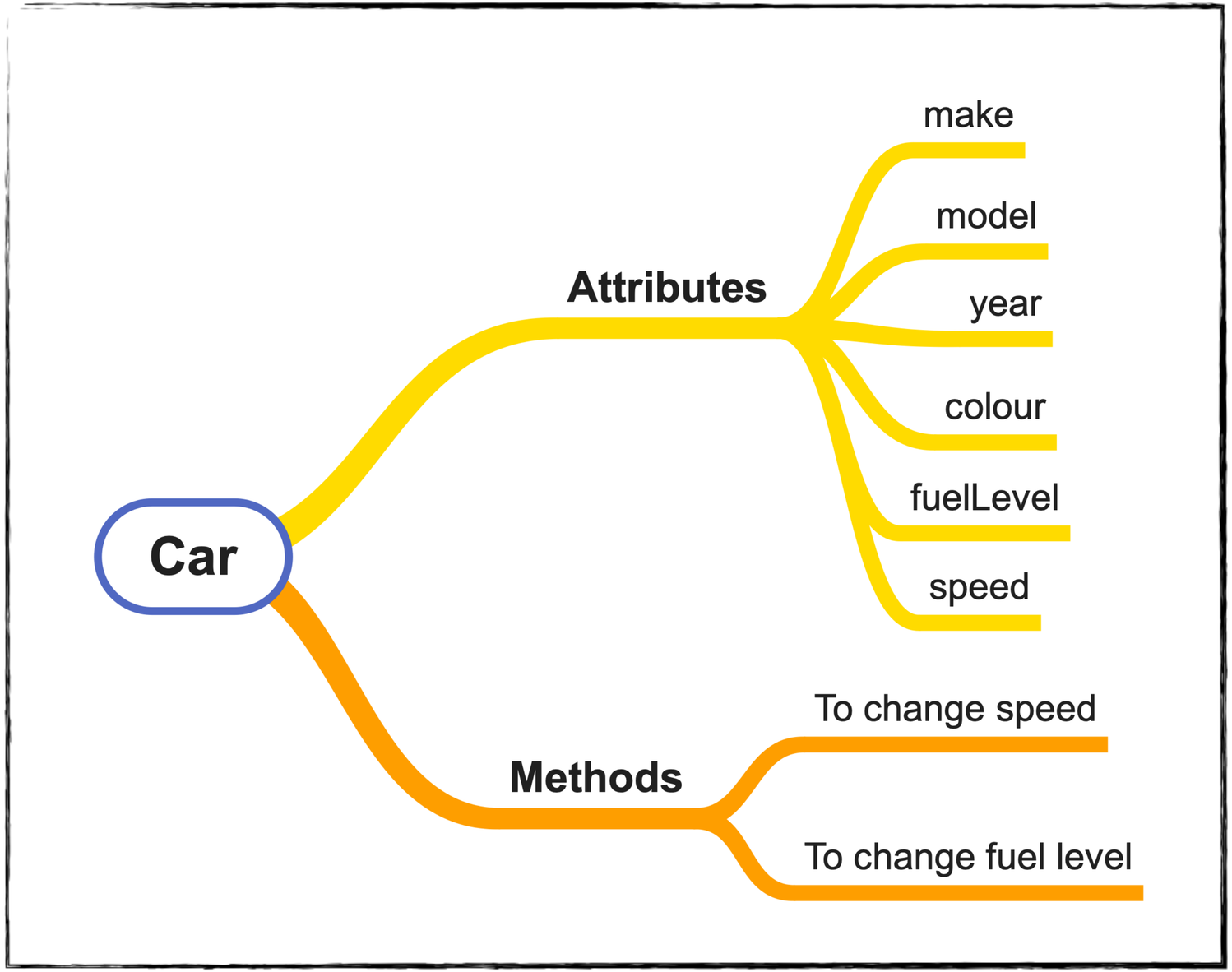
What is an Object?

- A collection of related information (data);
- Data inside an object called: class fields or instance variables;
- Operations inside the object are methods.

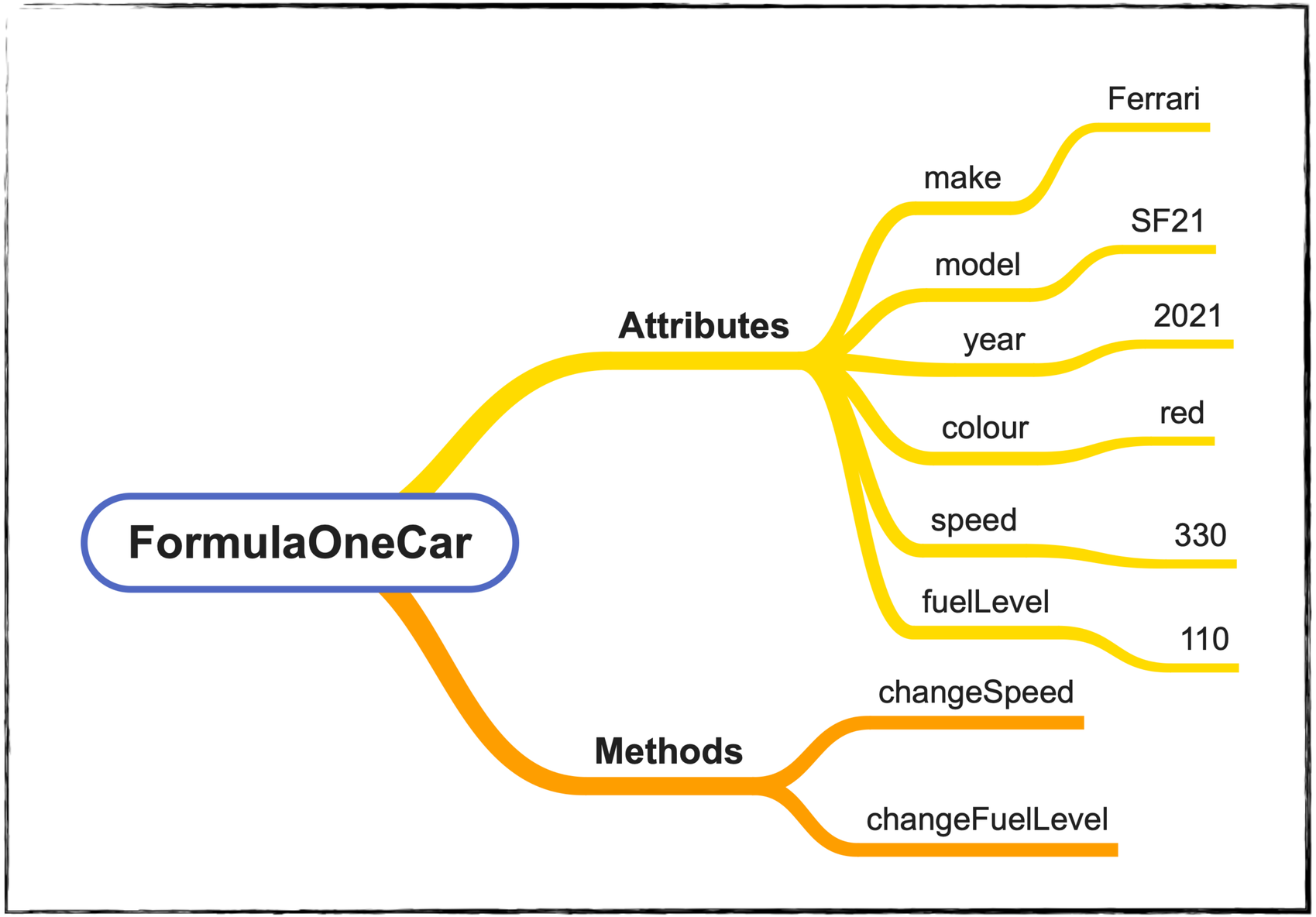
Classes vs. Objects

- A class describes what is in an object;
- An object is is created from the class specification;
- The house analogy would be:
  - Construction plans for a House are the equivalent of a class;
  - As many houses as required can be constructed from the plans;
  - Each house conforms to the plans in exactly the same way;
  - Each house is built in a different location;
  - Each house has the same structure, different things inside it;
  - House plan is the class, houses are the objects.

A Car Class


Car
Attributes
make
model
year
colour
fuelLevel
speed
Methods
To change speed
To change fuel level

A Car Object


FormulaOneCar
Attributes
make
Ferrari
model
SF21
year
2021
colour
red
speed
330
fuelLevel
110
Methods
changeSpeed
changeFuelLevel

Inheritance

- Facilitates the creation of class hierarchies;
- Create new classes built on existing classes;
- A child class inherits the parent class properties and behaviours;
- A class inherited from is a parent/super class; and
- A class that inherits from another is a subclass/child class.

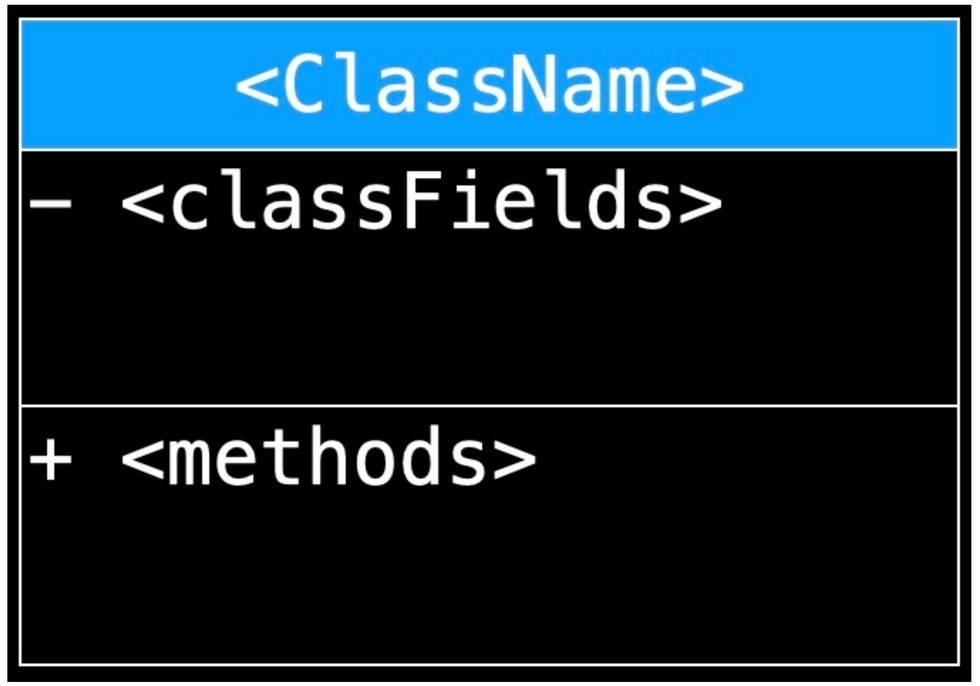
Inheritance Example

- A Person;
- An Employee is a Person;
- A Manager is an Employee who is a Person;
- A Person, an Employee and a Manager all have a name;
- A Person does not have a Staff ID;
- An Employee and a Manager have a Staff ID;
- An Employee does not have subordinates, a Manager does.

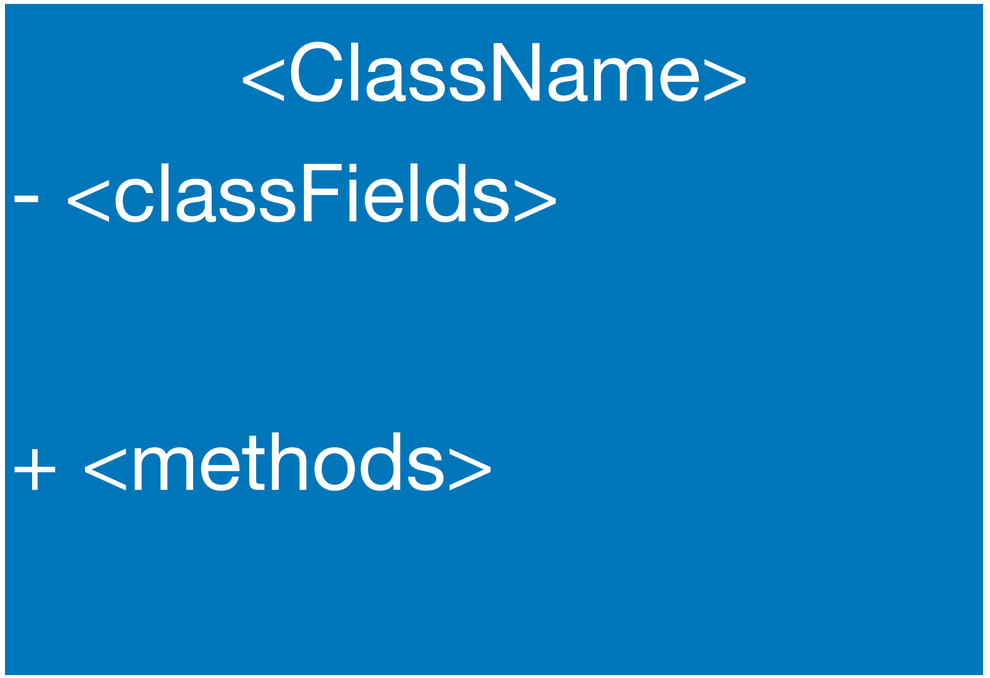
UML Class Diagrams

Designing a Class

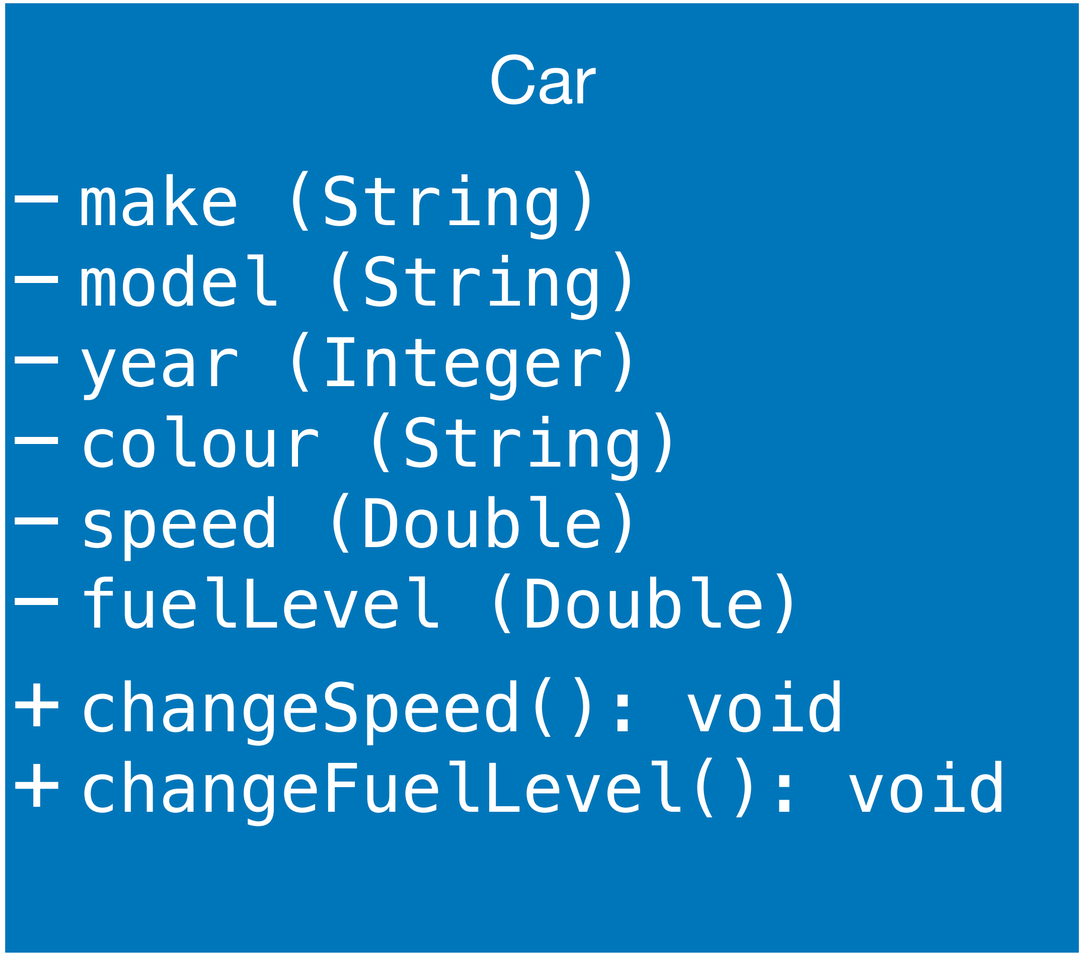
- The Unified Modelling Language (UML) used to draw classes;
- A class diagram is a rectangle with 3 sections:
  1. Top section: contains class name;
  2. Middle section: contains attributes' names and data types;
  3. Bottom section: contains the methods.



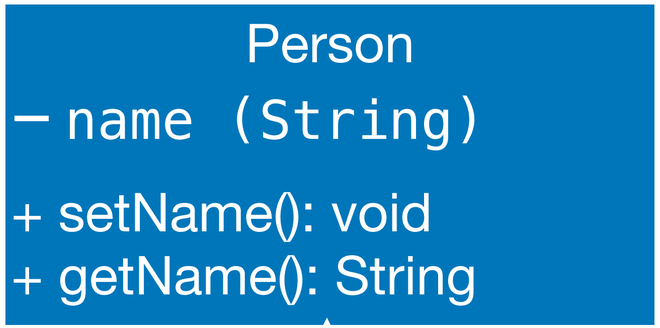
A UML Class Diagram Template

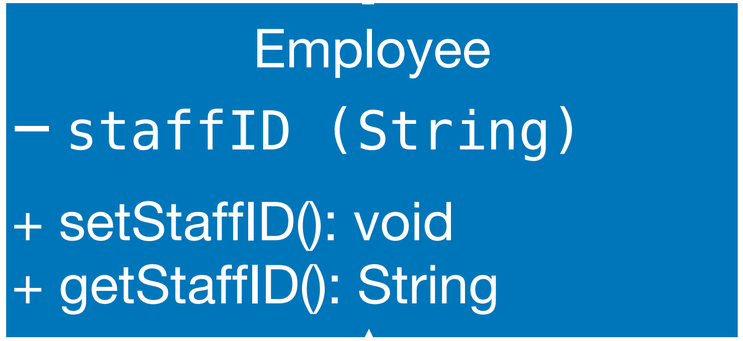


Car Class Diagram

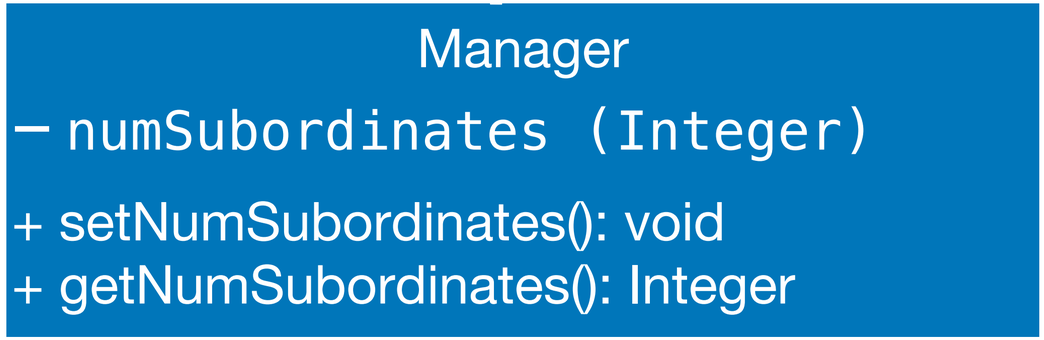


Inheritance Class Diagram





Employee has a name



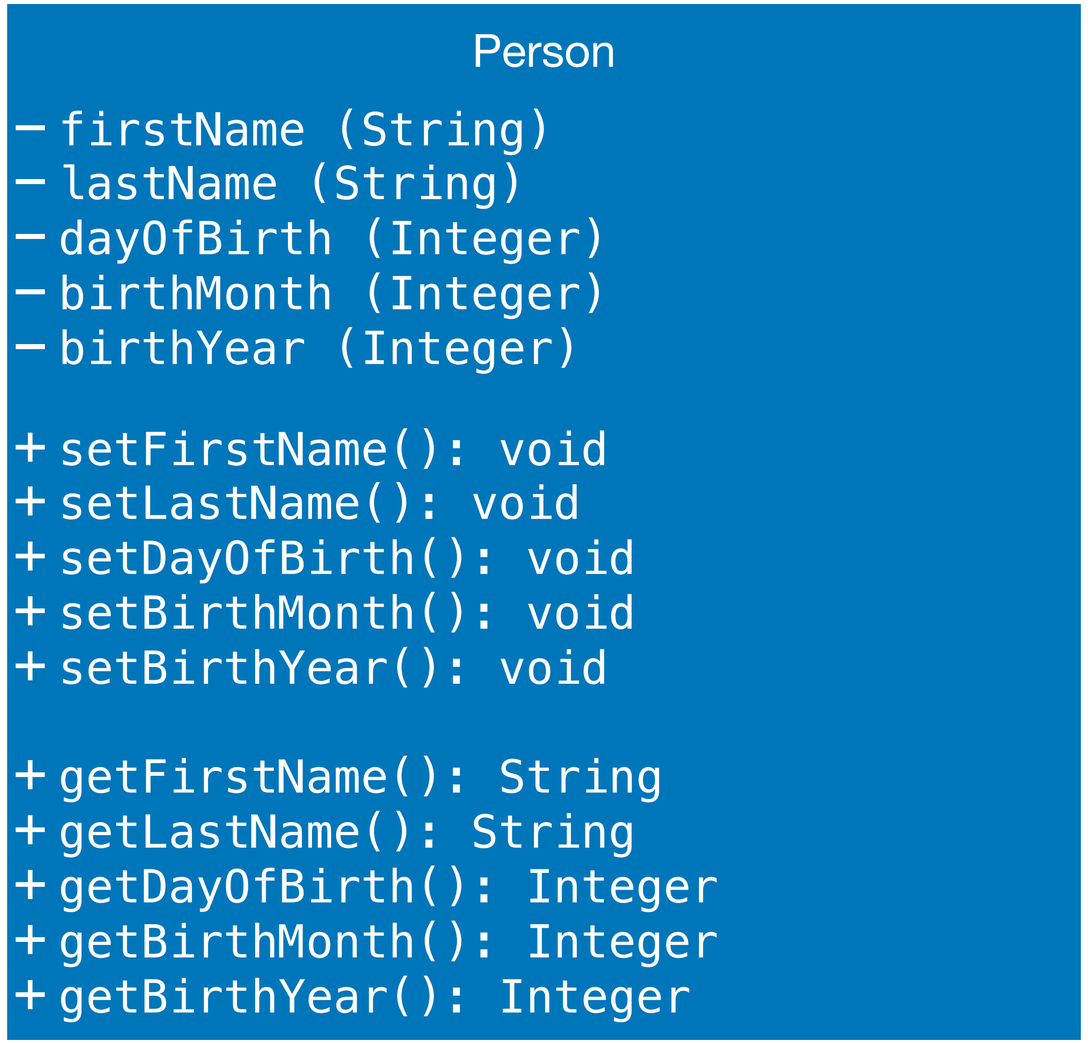
Manager has a name and has a staffID

Classes and Pseudocode

An Example – Person

- Our person has the following attributes:
  - a first name;
  - a family name;
  - a day of birth (dd);
  - a month of birth (mm); and
  - a year of birth (yyyy).
- A person has other attributes, this is enough for now.

Person Class UML Diagram



Class Fields in Pseudocode

- These are the variables that make up our person.

```
CLASS Person
      CLASS FIELDS:
            firstName (String)
            lastName (String)
            dayOfBirth (Integer)
            birthMonth (Integer)
            birthYear (Integer)
      ...
```

Inheritance in Pseudocode - Class Fields

```
CLASS Employee
      CLASS FIELDS:
          Inherits from Person
          staffID (String)
      ...
```

Accessors & Mutators

Working with Values in Instance Variables

- Things change, values change;
- Values within instance variables evolve and need to change;
- Values within instance variables must be retrieved for use;
- Accessors (Getters) and Mutators (Setters) are class methods facilitating this.

Accessor Methods

- AKA “Getters”;
- Used to retrieve information from the object;
- Clearly define the data types of the information to be retrieved from the object;
- Each Accessor method retrieves a single class field’s value;

Accessor Method in Pseudocode

```
ACCESSOR: getFirstName
IMPORT: none
EXPORT: firstName (String)
ASSERTION: Returned the first name
ALGORITHM:
    EXPORT COPY OF firstName

Alternate ALGORITHM:
    RETURN COPY OF firstName
```

- Note: **getLastName()**, **getDayOfBirth()** and **getBirthMonth()** and **getBirthYear()** are done similarly.

Mutator Methods

- AKA “Setters”
- Used to send information into the object which is used to modify the class field values;
- This new information changes the class fields’ values;
- It mutates (changes) the object’s state;
- Clearly define the data types of the information to be passed into the object;
- There is usually a mutator method for each class field.

Mutators and IMPORT Validation

- If possible, the IMPORT used must be validated:
  - If it is valid, then it is to be used to update the object state;
  - If it is invalid then it is not used:
    - The mutation is not permitted;
    - Throw an exception, Inform the user...

Mutators and IMPORT Validation (2)

- If the IMPORT is validated, the software system can always assume each object has a valid state;
- Of course, sometimes validation is not possible:
  - Person's name;
  - Title of a book.
- Generally check null for Strings.

Validation, Validation...

- Absolutely every piece of data must be validated;
- Validating dates: what dates are valid when?
- Validation may require more methods within a Class.

Mutator Methods - Pseudocode

```
MUTATOR: setFirstName
IMPORT: pFirstName (String)
EXPORT: none
ASSERTION: State of firstName is updated to pFirstName value
ALGORITHM:
    firstName = pFirstName
```

- Note: **setFamilyname()**, **setDayOfBirth()** and **setBirthMonth()** and **setBirthYear()** are done similarly;
- Why might some validation be needed?

Mutator Methods - Validation Pseudocode

- We have created a class called Rectangle;
- **sideLength** and **sideHeight** are two variables;
- Let's validate one of them (the other would be done similarly):

```
MUTATOR: setSideLength
IMPORT: pSideLength (double)
EXPORT: none
ASSERTION: State of sideLength is updated to pSideLength value
ALGORITHM:
    IF pSideLength <= 0 THEN
        ERROR "Side length must have a positive value"
    ELSE
        sideLength = pSideLength
    ENDIF
```

Example – The State of Two Person Objects

Object personOne state:

firstName = "Mohamed;
familyName = "Sallah";
dayOfBirth = 15;
birthMonth = 6;
birthYear = 1992;

Object personTwo state:

firstName = "Jordan";
familyName = “Henderson";
dayOfBirth = 17;
birthMonth = 6;
birthYear = 1990;