Object-Oriented Programming

Notes compiled by:

Neelam Pirbhai-Jetha

Notes on:

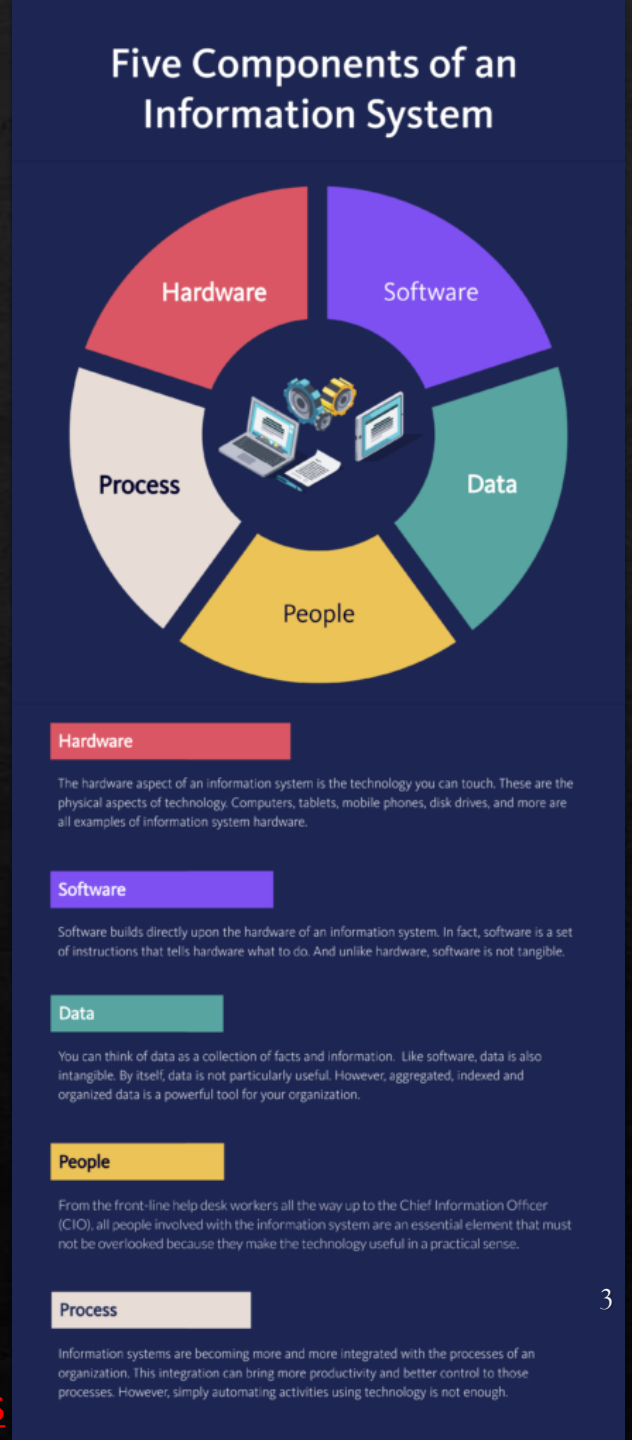
Unit 1: Introduction to Information System

Unit 2: Information Systems and their importance

Unit 3: Object-oriented Information Systems

Unit 1: Introduction to Information System
&
Unit 2: Information Systems and their importance

- Introduction:
- "Information systems are combinations of hardware, software, and telecommunications networks that people build and use to collect, create, and distribute useful data, typically in organizational settings" (cited in Bourgeois, 2014: 5).
- Information systems have five major components:
  - hardware,
  - software,
  - data,
  - people,
  - process



Source: Components

Unit 1 & Unit 2

- "Information systems have a number of vital components, some tangible, others intangible, and still others of a personnel nature. These components collect, store, organize, and distribute data throughout the organization" (Bourgeois, 2014:8)
- Furthermore, "one of the roles of information systems is to take data and turn it into information, and then transform that information into organizational knowledge" (Ibid).
- Concepts surrounding the notion of Information Systems are:
  - data,
  - information,
  - knowledge.
- DATA + MEANING = INFORMATION
- i.e Information=data in context
- Knowledge can be viewed as information that facilitates action.
- There are different types of knowledge:
  - explicit knowledge (e.g learning by heart, remembering facts)
  - tacit knowledge (e.g. "knowledge that would be difficult to pass to another person simply by writing it down“ (Buckley, 2021)
- References:

- Bourgeois, D. (2014) *Information Systems for Business and Beyond*. Saylor Academy.
- Buckley, O. (2021) Unit 1: Introduction to Information Systems, Lecture Notes, Object-oriented Information Systems OOIS_PCOM7E, University of Essex Online, delivered August 2021.

Unit 3: Object-Oriented Information Systems

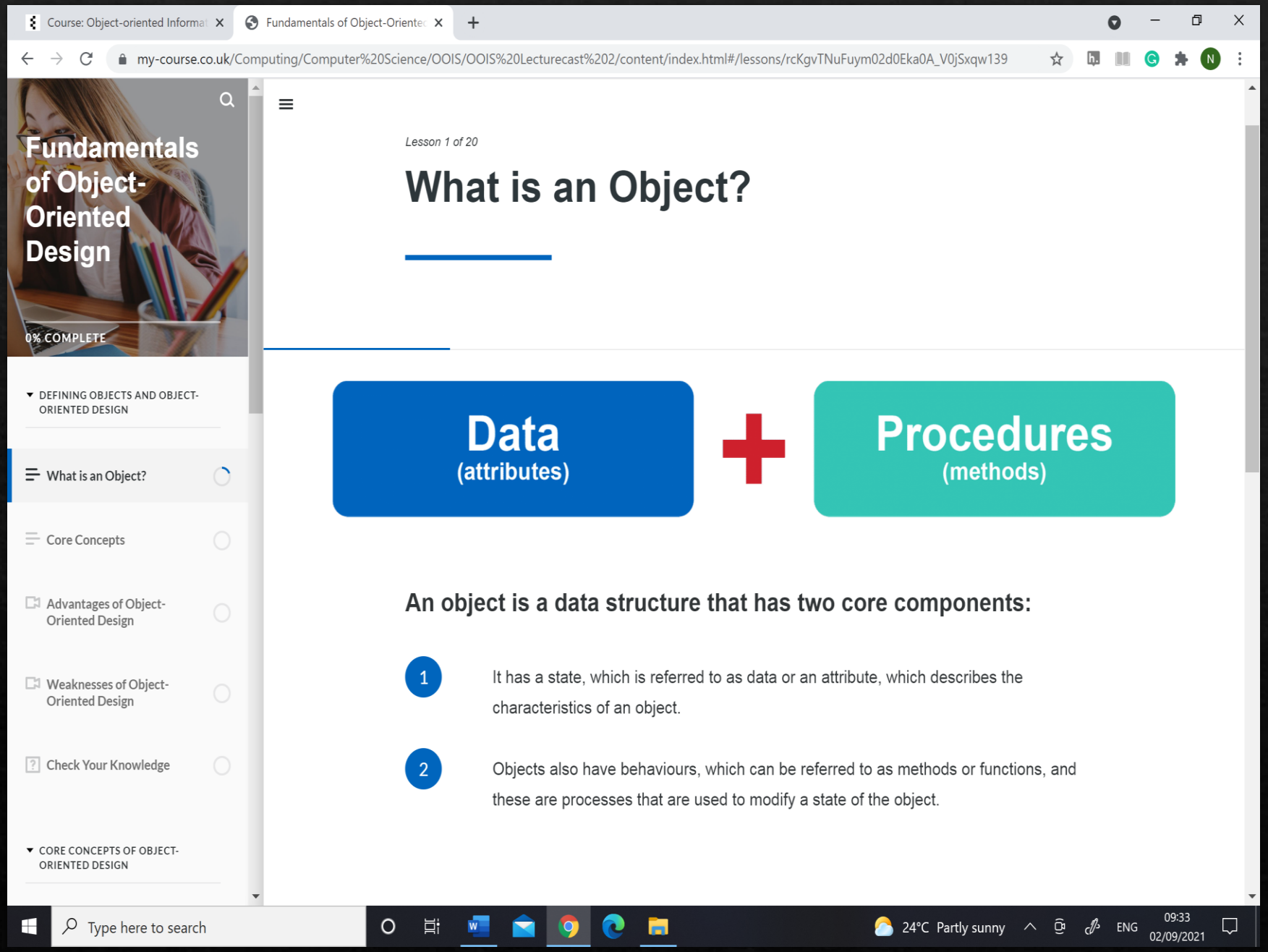
- **In this unit, I will learn how to:**
- Identify the appropriate objects within a system,
- Develop an object-oriented design for a system,
- Correctly apply composition and inheritance where appropriate.

◈ If not specifically mentioned, all notes are taken from:

◈ Philips, D. (2018) « Chapter 1: Object-Oriented Design », *Python 3 Object-Oriented Programming*, Birmingham: Packt Publication. 3rd Edition.

Preliminary Notes

- In software development, design is often considered as the step done before programming.
- But, in reality, analysis, programming, and design tend to overlap, combine, and interweave.



Source: Buckley, O. (2021)

i)Objects will have state (fields, attributes, variables). Fields or state are used to describe the internal state of the object. Ex. Age, breed are states

ii)The methods or behaviours are how you get your object to do something – typically the way that you change the internal state of an object is to use a method to modify it. Ex. going back to the cat example, you might have a method or behaviour called eating that then updates the hungry state.

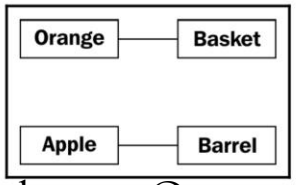
OBJECT

- Object = a tangible thing that we can sense, feel, and manipulate. (General Definition)
- Software objects may not be tangible things that you can pick up, sense, or feel, but they are models of something that can do certain things and have certain things done to them. (software development definition)
- An object = a collection of data and associated behaviors.
- Oriented = directed toward.
- So object-oriented means functionally directed toward modeling objects. This is one of many techniques used for modeling complex systems. It is defined by describing a collection of interacting objects via their data and behavior.
- Object-oriented analysis (OOA) is the process of looking at a problem, system, or task (that somebody wants to turn into an application) and identifying the objects and interactions between those objects. The analysis stage is all about what needs to be done.
- [p. 7]

Understanding the terms: Objects vs Class

- P. 8: Example for creating an inventory application for a fruit farm is used.
- Apples and oranges are both objects. To facilitate the example, we can assume that apples go in barrels and oranges go in baskets.
- Now, we have four kinds of objects: apples, oranges, baskets, and barrels.
- In object-oriented modeling,

The relationship between the four classes of objects in our inventory system can be described using a **Unified Modeling Language** (invariably referred to as **UML**, because three-letter acronyms never go out of style) class diagram. Here is our first class diagram:

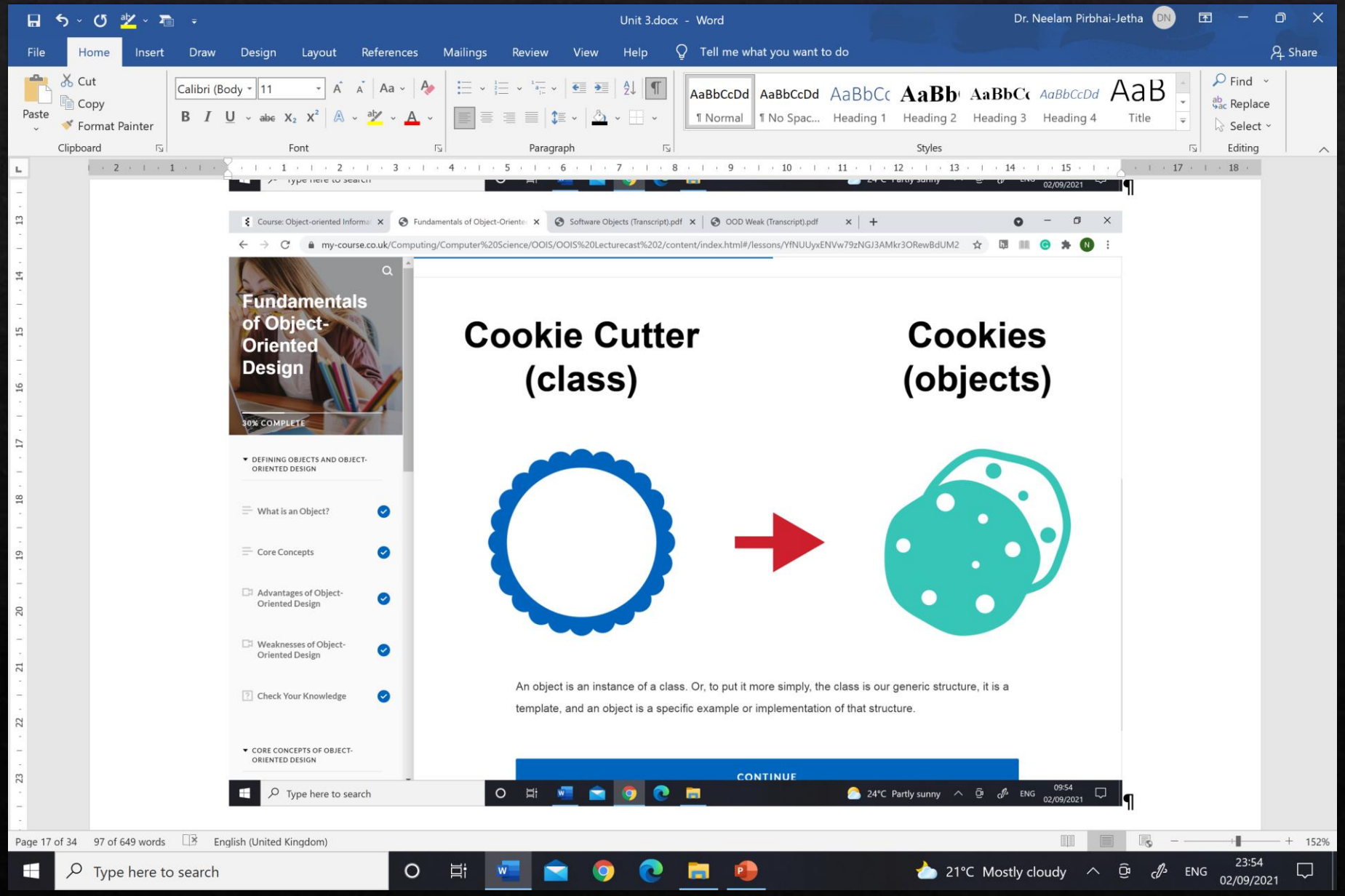


This diagram shows that an Orange is somehow associated with a Basket and that an Apple is also somehow associated with a Barrel. Association is the most basic way for two classes to be related.

class = a kind of object.

- Classes describe objects. They are like blueprints for creating an object. You might have three oranges sitting on the table in front of you. Each orange is a distinct object, but all three have the attributes and behaviors associated with one class: the general class of oranges. [p. 9]

Object vs Class



Source: Buckley, O. (2021)

Intro to Object Oriented Programming - Crash Course

What is OOP? - Classes

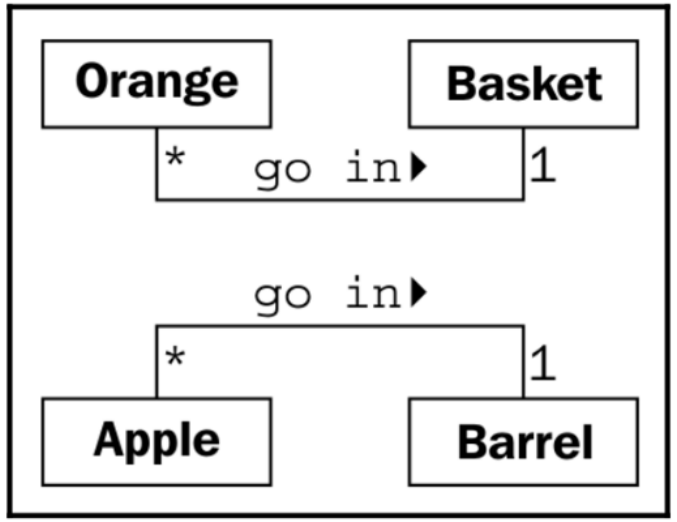
- Whereas the knight class represents any given knight, a knight object represents only one singular knight

7:08 / 30:17 • Introduction >

Source: https://www.youtube.com/watch?v=SiBw7os-_zI

p. 11 the option to add further clarification as needed.

The beauty of UML is that most things are optional.



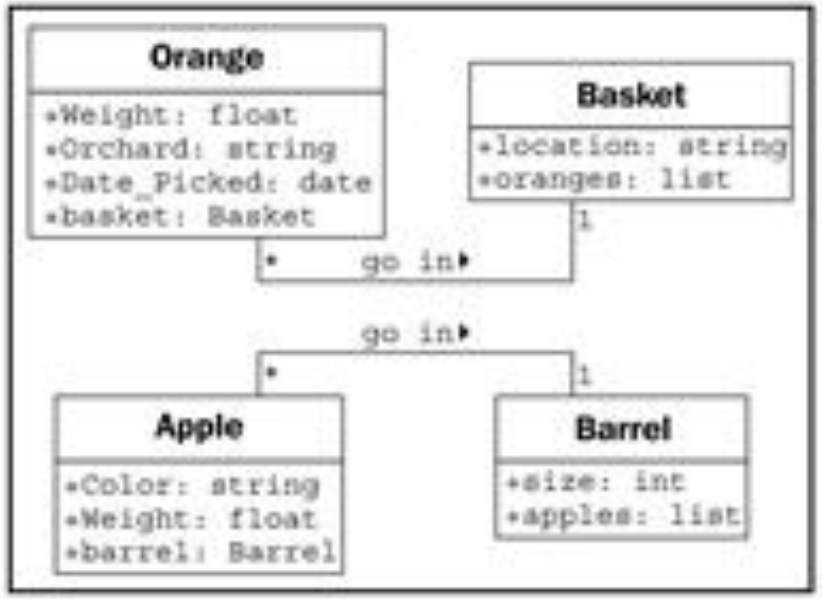
An object instance is a specific object with its own set of data and behaviors; a specific orange on the table in front of us is said to be an instance of the general class of oranges.

This diagram tells us that oranges **go in** baskets, with a little arrow showing what goes in what. It also tells us the number of that object that can be used in the association on both sides of the relationship. One **Basket** can hold many (represented by a *) **Orange** objects. Any one **Orange** can go in exactly one **Basket**. This number is referred to as the *multiplicity* of the object. You may also hear it described as the *cardinality*. These are actually slightly distinct terms. Cardinality refers to the actual number of items in the set, whereas multiplicity specifies how small or how large the set could be.

- p. 12 Data = individual characteristics of a certain object.
- A class can define specific sets of characteristics that are shared by all objects from that class.
- Any specific object can have different data values for the given characteristics.
- For example, the three oranges on our table could each weigh a different amount. The orange class could have a weight attribute to represent that datum.
- All instances of the orange class have a weight attribute, but each orange has a different value for this attribute.
- Attributes don't have to be unique, though; any two oranges may weigh the same amount. As a more realistic example, two objects representing different customers might have the same value for a first name attribute.

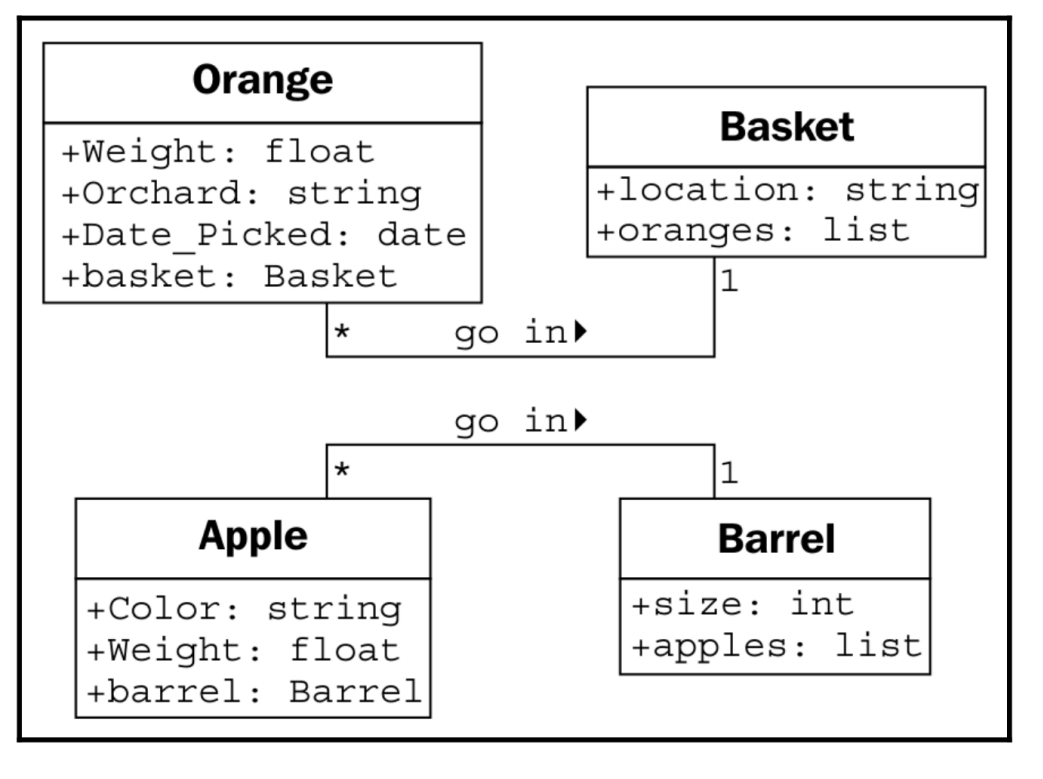
Adding a few different attributes to the class diagram:

Depending on how detailed our design needs to be, we can also specify the type for each attribute. A ttribute types are often primitives that are standard to most programming languages, such as integer, floating-point number, string, byte, or Boolean. However, they can also represent data structures such as lists, trees, or graphs, or most notably, other classes. This is one area where the design stage can overlap with the programming stage. The various primitives or objects available in one programming language may be different from what is available in another:

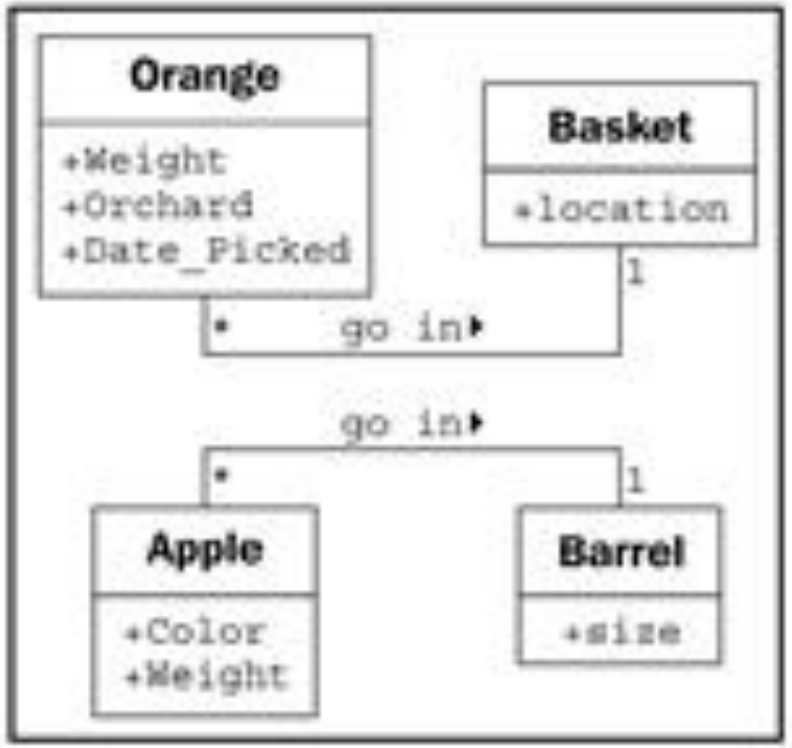


- Attributes are frequently referred to as members or properties.

Specify the type (float, integer...) for each attribute (to get a more detailed diagram - p. 13)



In our fruit inventory application, the fruit farmer may want to know what orchard the orange came from, when it was picked, and how much it weighs. They might also want to keep track of where each **Basket** is stored. Apples might have a color attribute, and barrels might come in different sizes. Some of these properties may also belong to multiple classes (we may want to know when apples are picked, too), but for this first example, let's just add a few different attributes to our class diagram:



Data & Behaviours

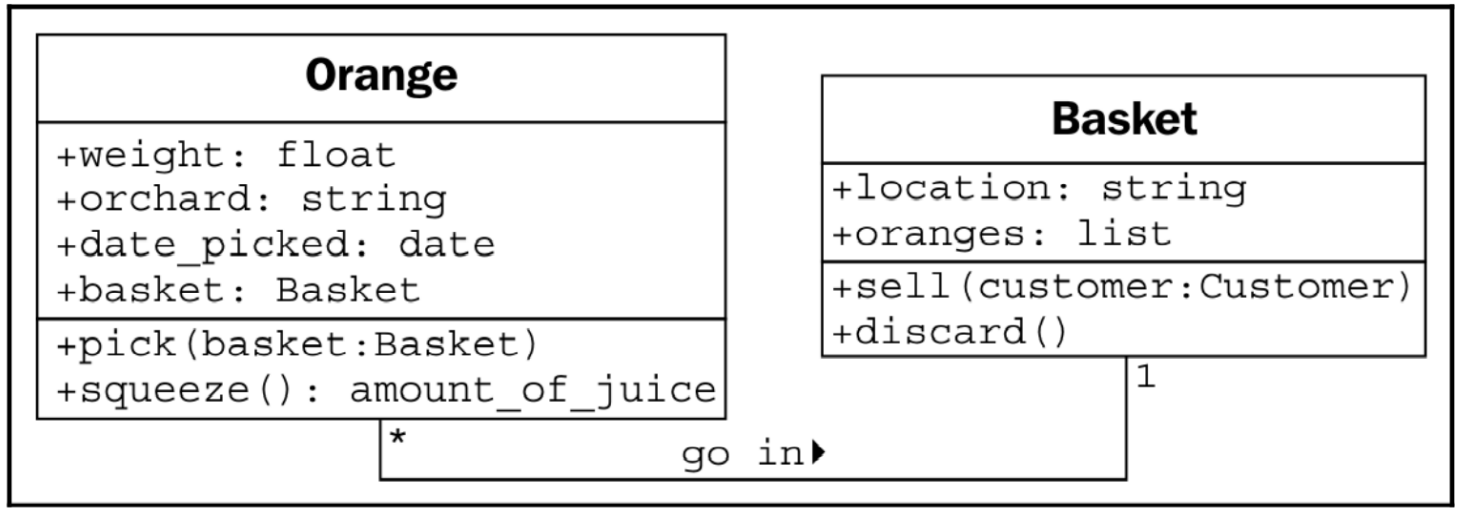
- Recap: Data = individual characteristics of a certain object.
- p.13-14
- Behaviors = actions that can occur on an object.
- The behaviors that can be performed on a specific class of object are called methods.
- At the programming level, methods are like functions in structured programming, but they magically have access to all the data associated with this object.
- Like functions, methods can also accept parameters and return values
- The actual object instances that are passed into a method during a specific invocation are usually referred to as arguments.
- These objects are used by the method to perform whatever behavior or task it is meant to do. Returned values are the results of that task.

inventory application. Let's stretch it a little further and see whether it breaks. One action that can be associated with oranges is the **pick** action. If you think about implementation, **pick** would need to do two things:

- Place the orange in a basket by updating the **Basket** attribute of the orange
- Add the orange to the **Orange** list on the given **Basket**.

So, **pick** needs to know what basket it is dealing with. We do this by giving the **pick** method a **Basket** parameter. Since our fruit farmer also sells juice, we can add a **squeeze** method to the **Orange** class. When called, the **squeeze** method might return the amount of juice retrieved, while also removing the **Orange** from the **Basket** it was in.

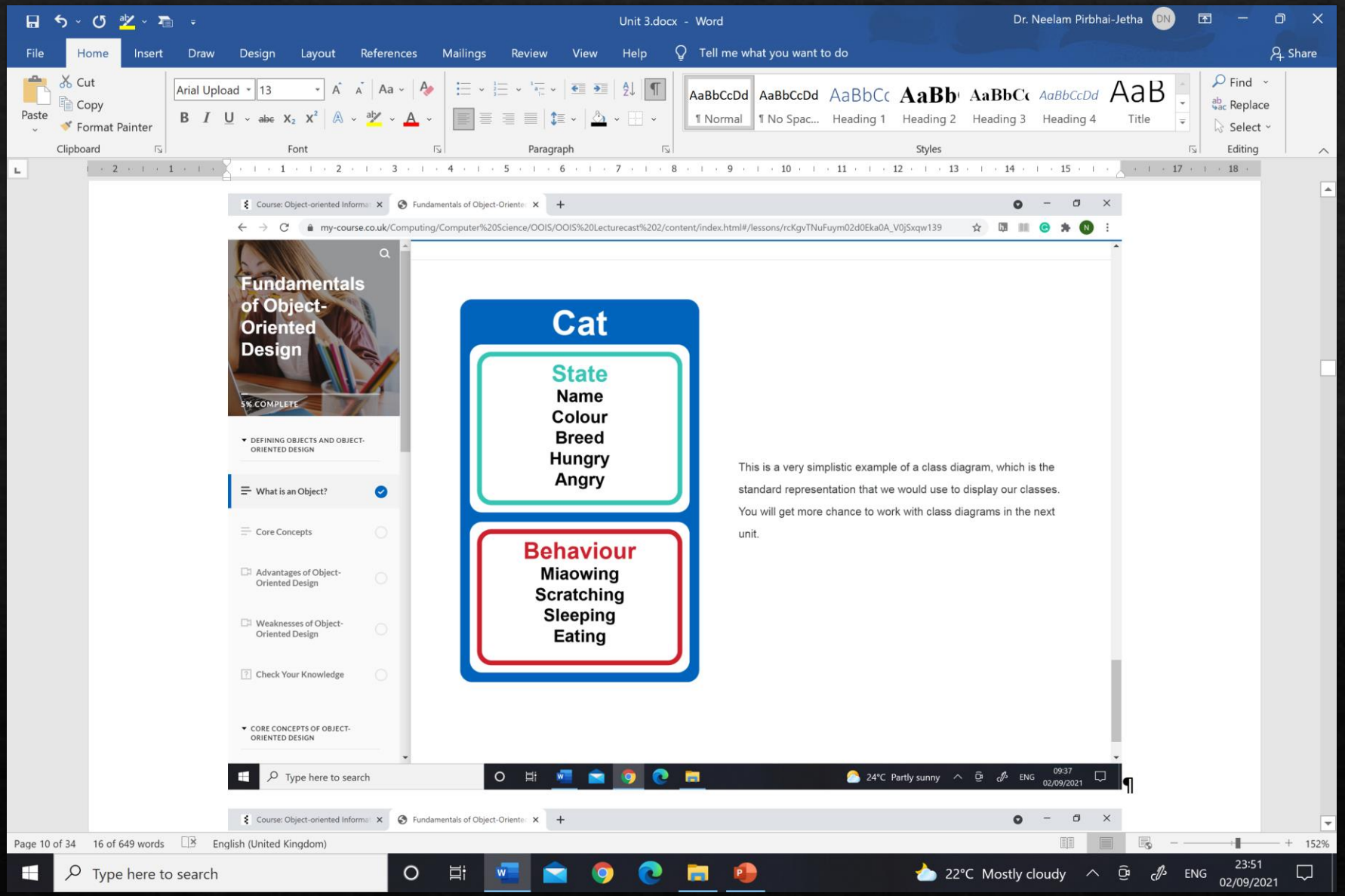
The class **Basket** can have a **sell** action. When a basket is sold, our inventory system might update some data on as-yet unspecified objects for accounting and profit calculations. Alternatively, our basket of oranges might go bad before we can sell them, so we add a **discard** method. Let's add these methods to our diagram:



p. 15 Object-oriented analysis and design is all about figuring out what those objects are and how they should interact.

Object has a state. behaviours, or functions or methods, that are used to change or update an object's state.

a state is usually associated with a noun
behaviours are typically associated with verbs

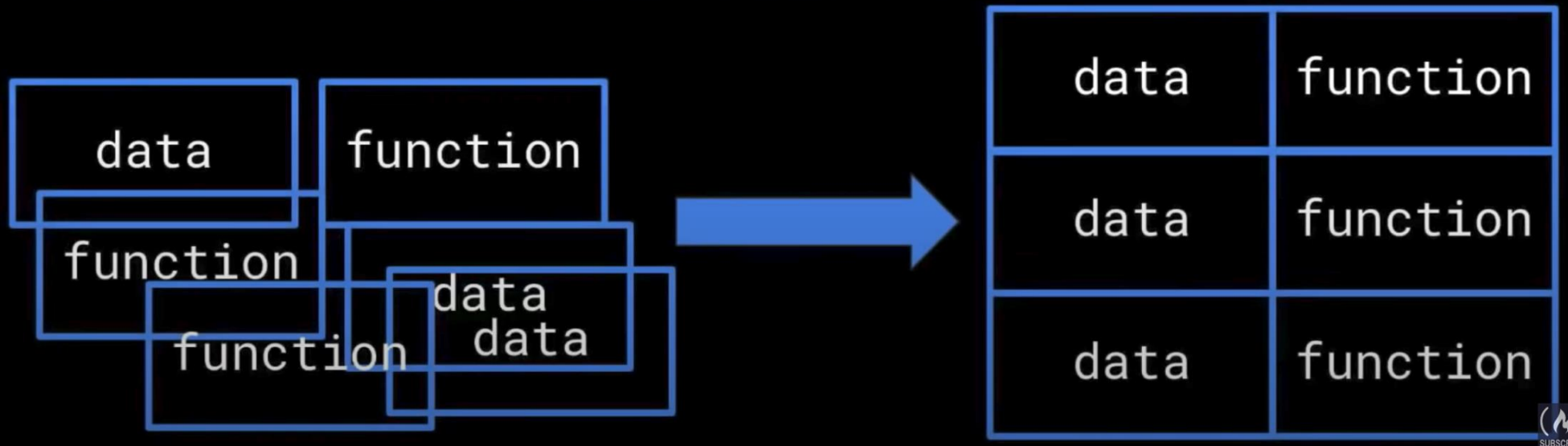


Source: Buckley, O. (2021)

Intro to Object Oriented Programming - Crash Course

What is OOP? - Classes

- **Object oriented programming** helps programmers create complex programs by **grouping together related data and functions**



7:19 / 30:17 • Introduction ›

Source: https://www.youtube.com/watch?v=SiBw7os-_zI

Basic Principles of OOP

- Types of relationships between objects:
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

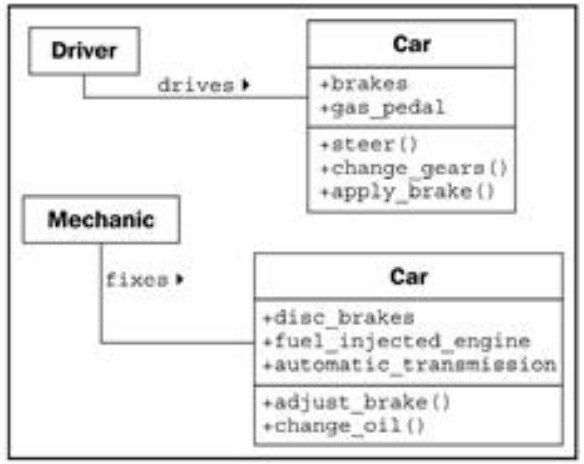
Interface (p. 15)

- The key purpose of modeling an object in object-oriented design is to determine what the public interface of that object will be.
- The interface is the collection of attributes and methods that other objects can access to interact with that object.
- They do not need, and are often not allowed, to access the internal workings of the object.
- The public interface, is very important + needs to be carefully designed as it is difficult to change it in the future.
- Must be kept simple
- Eg. interface to the television = the remote control

i)Encapsulation, ii)Abstraction

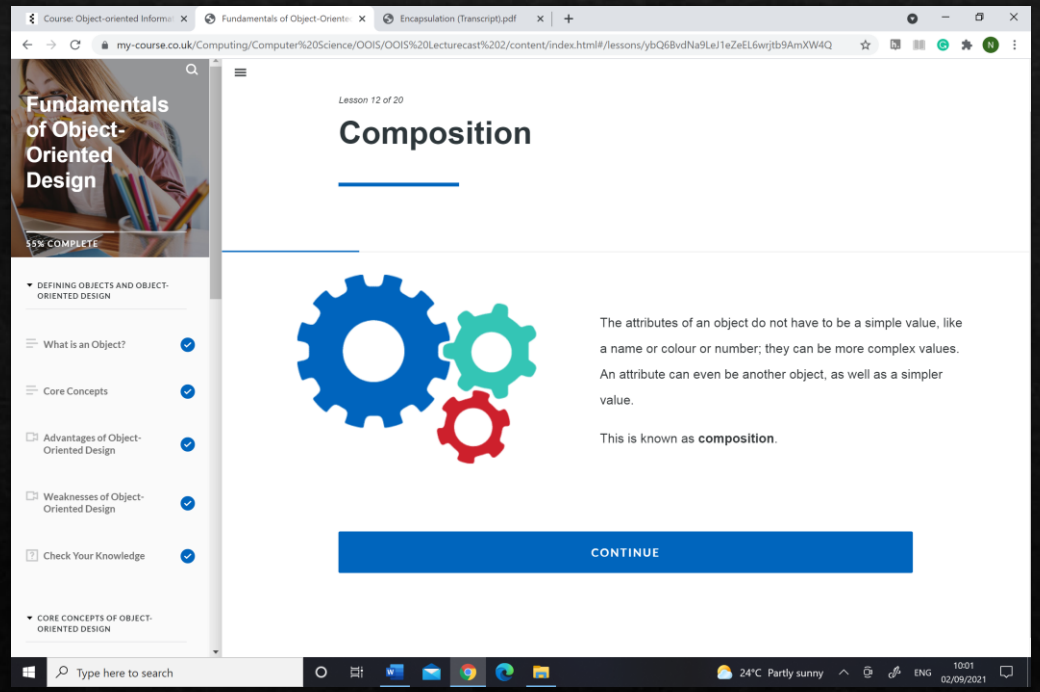
- process of hiding the implementation of an object is suitably called information hiding. It is also sometimes referred to as encapsulation.
- Abstraction is another object-oriented term related to encapsulation and information hiding.

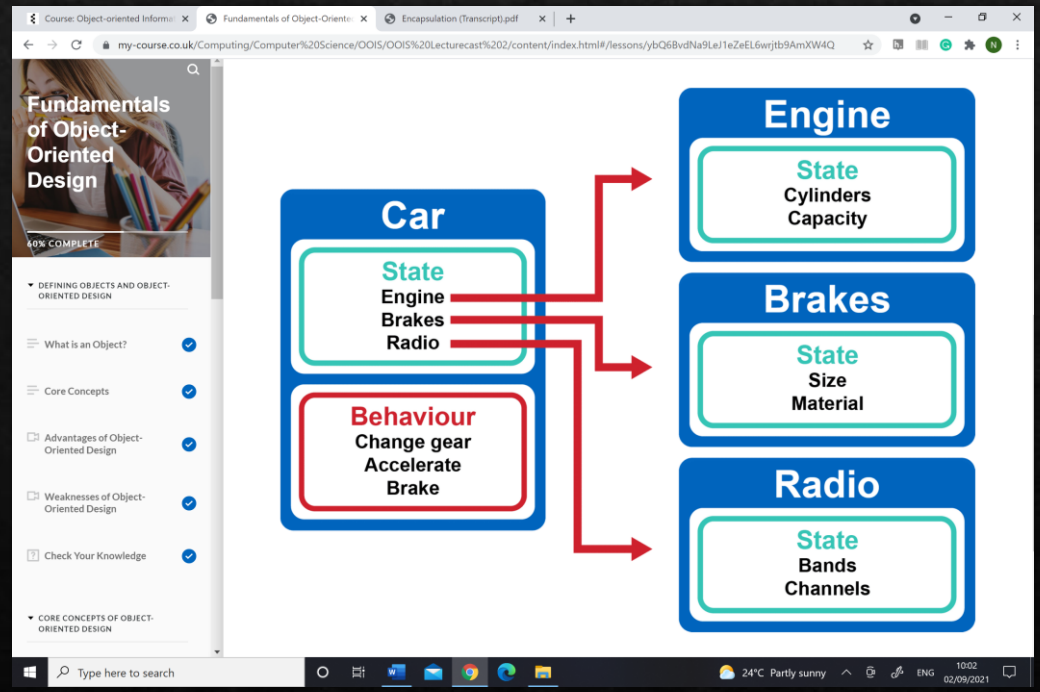
Abstraction is another object-oriented term related to encapsulation and information hiding. Abstraction means dealing with the level of detail that is most appropriate to a given task. It is the process of extracting a public interface from the inner details. A car's driver needs to interact with the steering, accelerator, and brakes. The workings of the motor, drive train, and brake subsystem don't matter to the driver. A mechanic, on the other hand, works at a different level of abstraction, tuning the engine and bleeding the brakes. Here's an example of two abstraction levels for a car:



- So far, we have learned to design systems as a group of interacting objects, where each interaction involves viewing objects at an appropriate level of abstraction.
- most design patterns rely on two basic objectoriented principles known as composition and inheritance
- Composition is the act of collecting several objects together to create a new one. Composition is usually a good choice when one object is part of another object. [p. 17]

Composition





Aggregation vs Composition

- P. 19. Aggregation is almost exactly like composition. The difference is that aggregate objects can exist independently.
- Another way to differentiate between aggregation and composition is to think about the lifespan of the object.
- If the composite (outside) object controls when the related (inside) objects are created and destroyed, composition is most suitable. If the related object is created independently of the composite object, or can outlast that object, an aggregate relationship makes more sense. Also, keep in mind that composition is aggregation; aggregation is simply a more general form of composition. Any composite relationship is also an aggregate relationship, but not vice versa

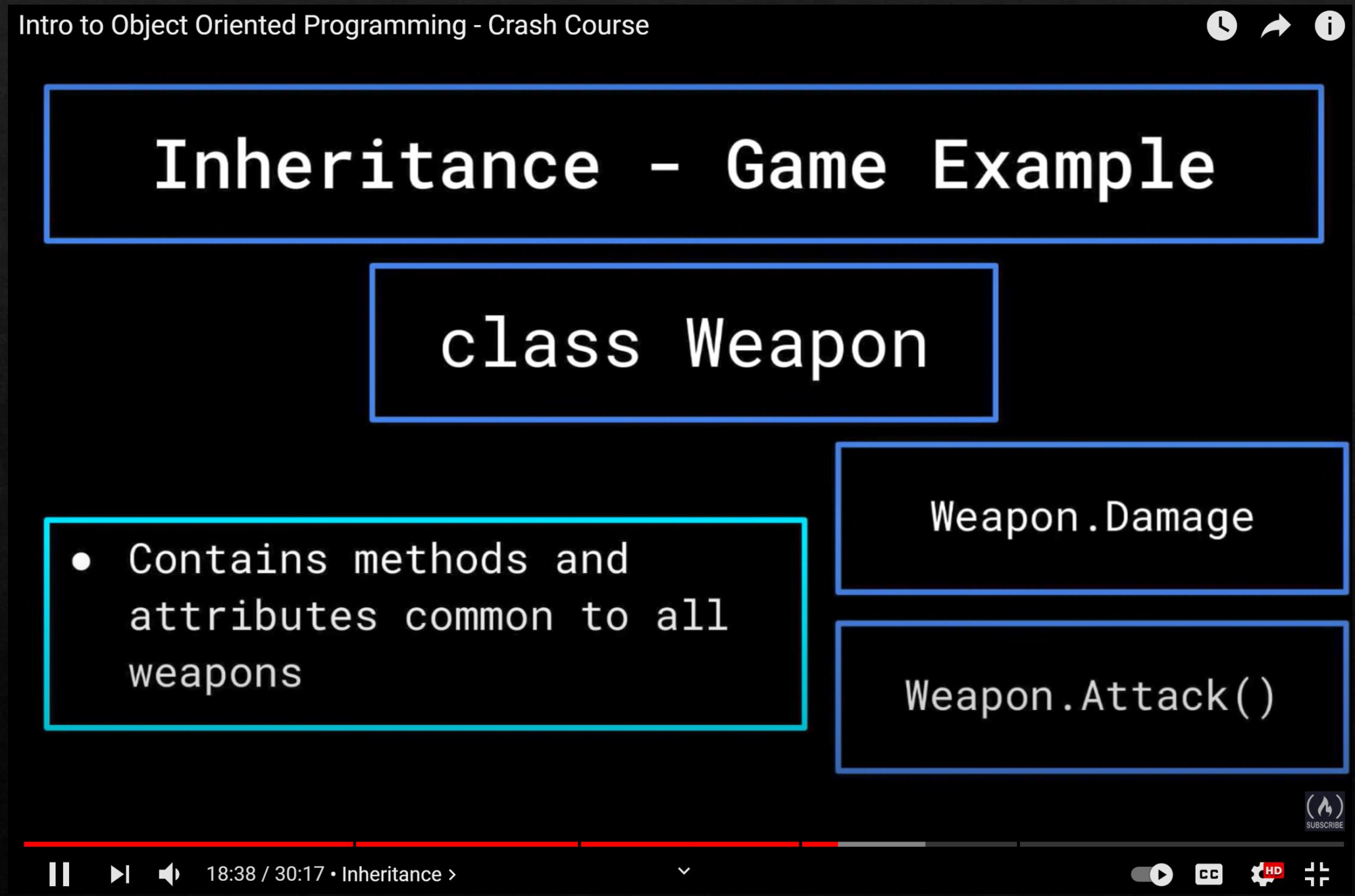
Inheritance

- *is a* = a cornerstone of inheritance

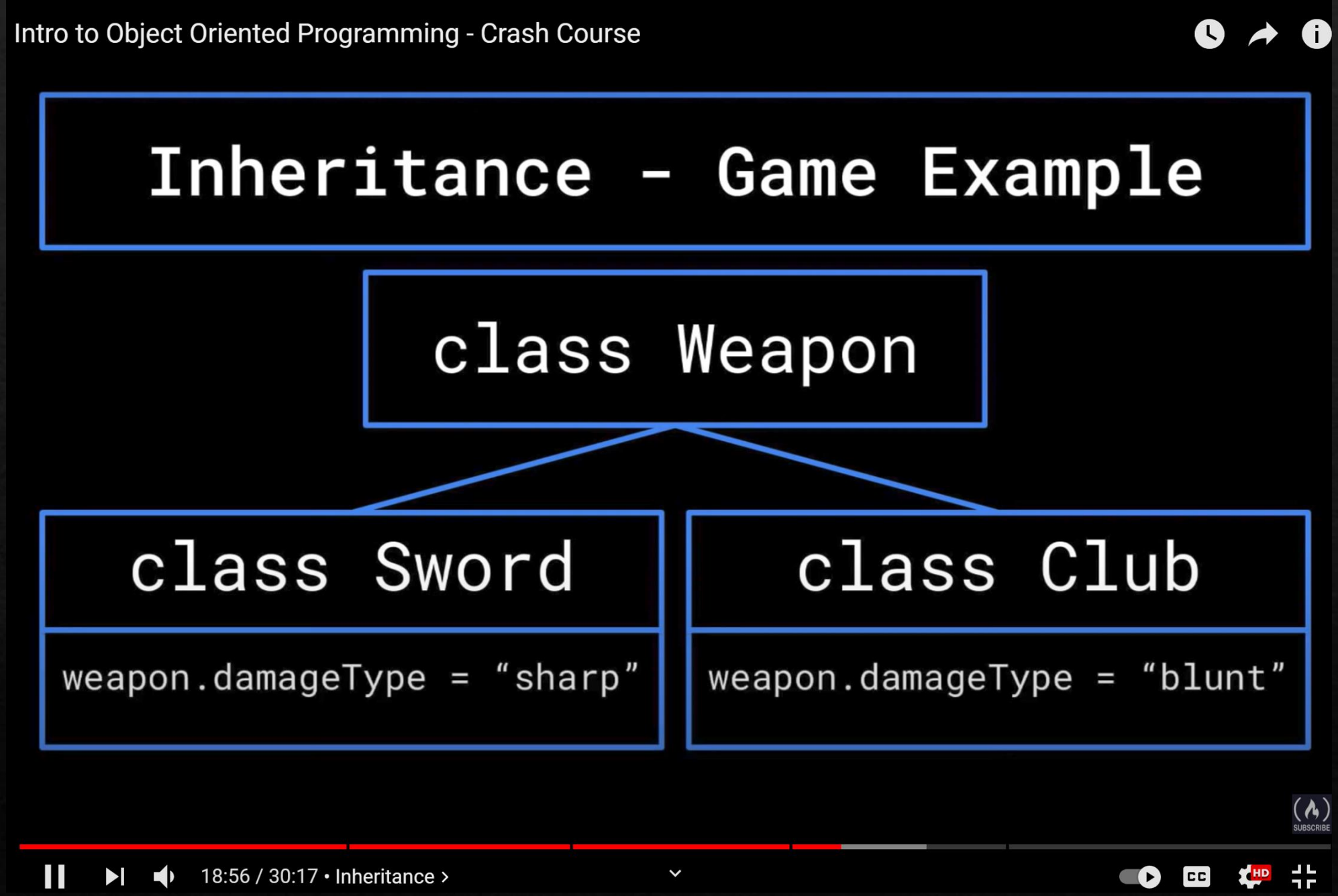
```
• Inheritance is the principle that allows
  classes to derive from other classes.
```

- Object-oriented design can also feature such multiple inheritance, which allows a subclass to inherit functionality from multiple parent classes. In practice, multiple inheritance can be a tricky business, and some programming languages (most famously, Java) strictly prohibit it.

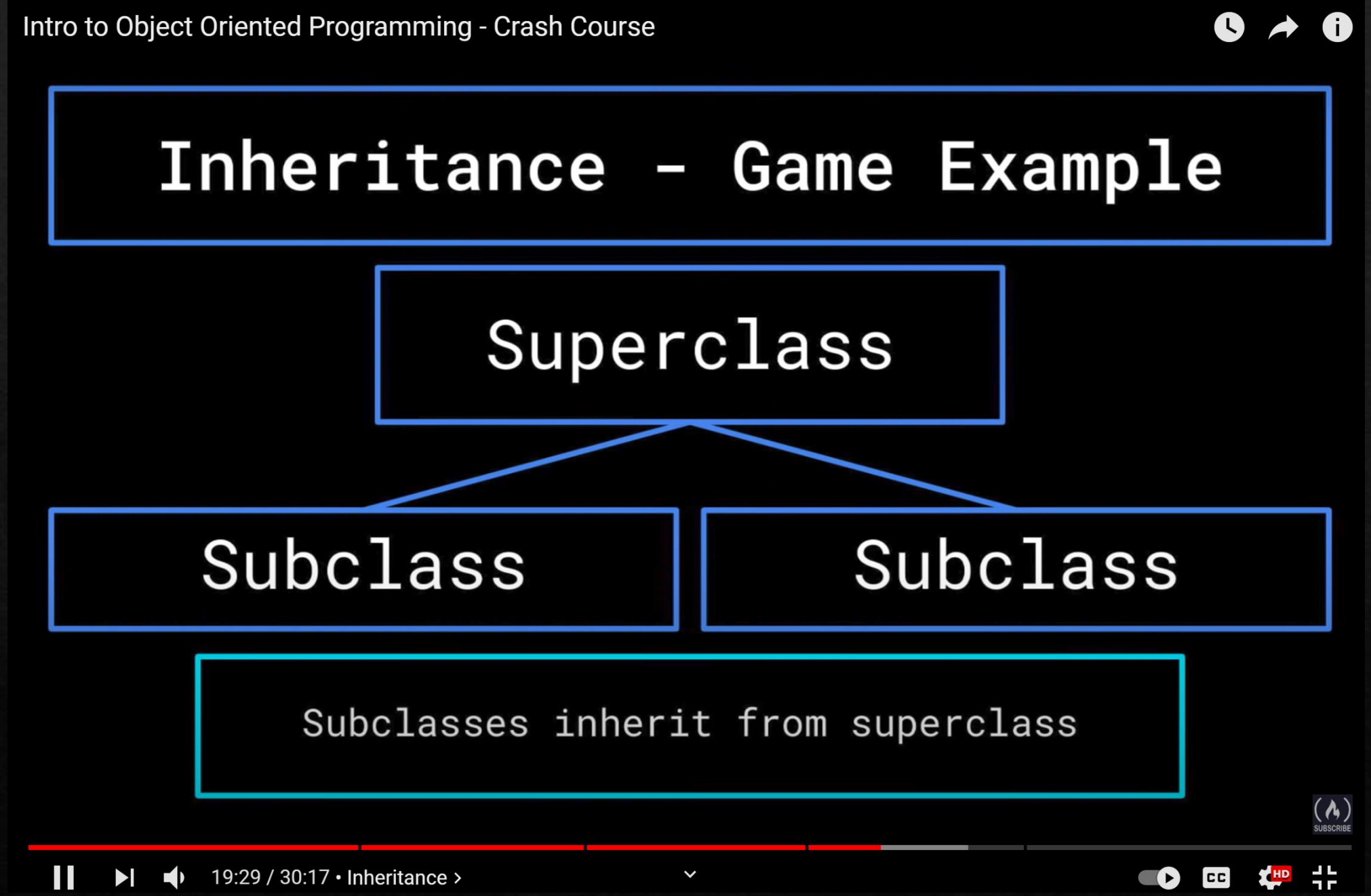
Source: https://www.youtube.com/watch?v=SiBw7os-_zI



Source: https://www.youtube.com/watch?v=SiBw7os-_zI



Source: https://www.youtube.com/watch?v=SiBw7os-_zI



Source: https://www.youtube.com/watch?v=SiBw7os-_zI

Unit 3.docx - Word

Dr. Neelam Pirbhai-Jetha

File Home Insert Draw Design Layout References Mailings Review View Help Tell me what you want to do Share

Cut Copy Format Painter Paste Clipboard

Calibri (Body) 11 Font

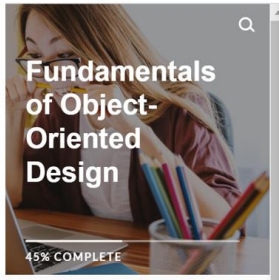
Paragraph

AaBbCcDd ¶ Normal | AaBbCcDd ¶ No Spac... | AaBbCc Heading 1 | AaBb Heading 2 | AaBbCc Heading 3 | AaBbCcDd Heading 4 | AaB Title Styles

Find Replace Select Editing

Course: Object-oriented Informa | Fundamentals of Object-Oriente | Encapsulation (Transcript).pdf

my-course.co.uk/Computing/Computer%20Science/OOIS/OOIS%20Lecturecast%202/content/index.html#/lessons/c8Ksl0dpmzskStqgn82kCUvbyAxxe5x2

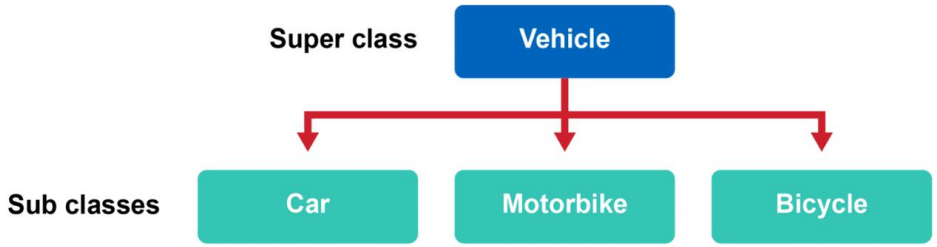


DEFINING OBJECTS AND OBJECT-ORIENTED DESIGN

- What is an Object?
- Core Concepts
- Advantages of Object-Oriented Design
- Weaknesses of Object-Oriented Design
- Check Your Knowledge

CORE CONCEPTS OF OBJECT-ORIENTED DESIGN

The class that is inherited from is commonly referred to as the super class and it is effectively a parent to the subclasses that inherit from it.



In this example, based on the vehicles that we have previously discussed, it is easy to see that a vehicle is our generic structure (our super class) and that a car, motorbike and bicycle are all types of vehicle that inherit from our super class.

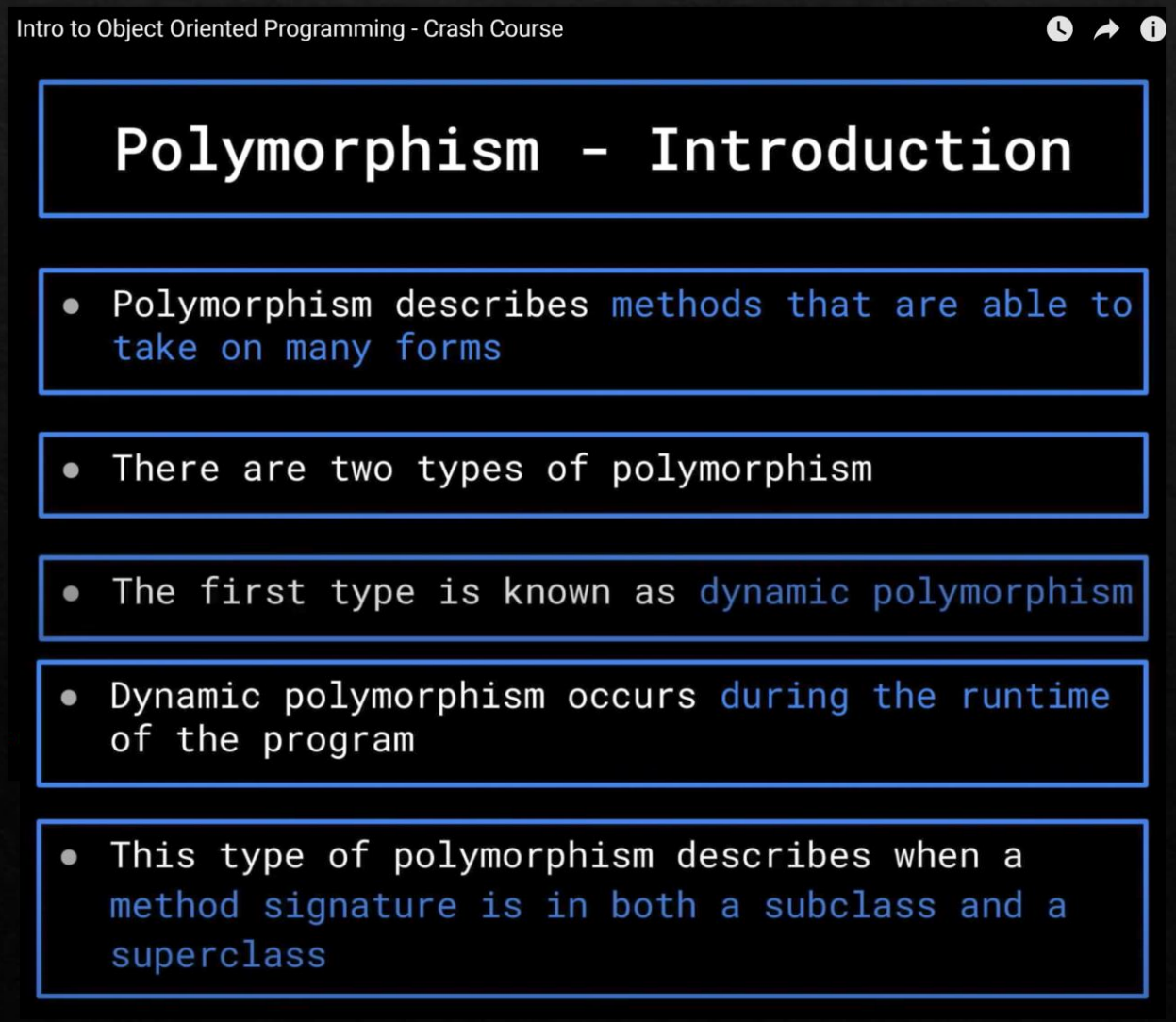
CONTINUE

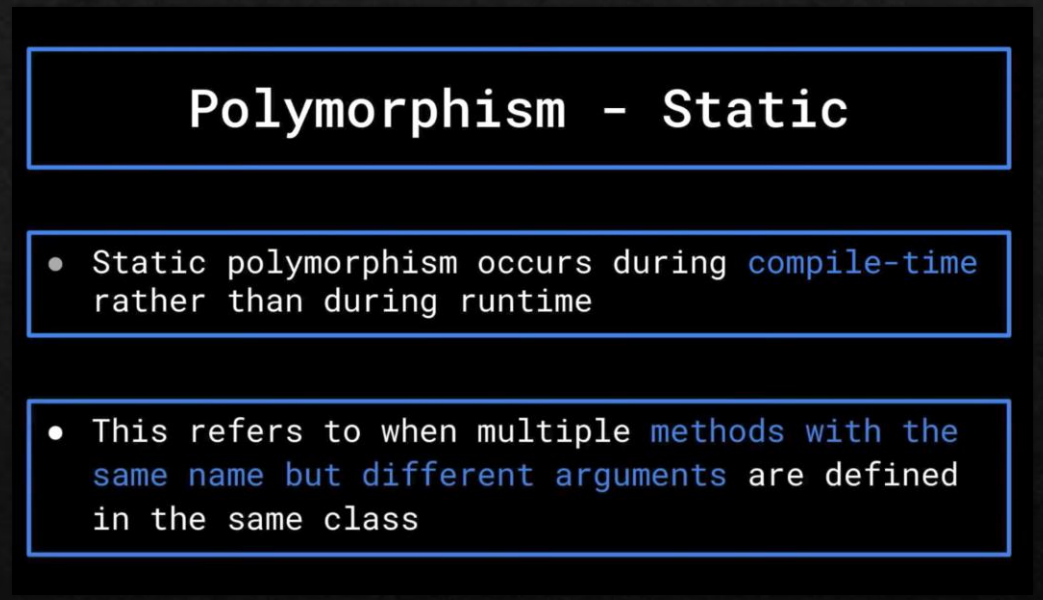
Type here to search 24°C Partly sunny Show hidden icons ENG 09:59 02/09/2021

Page 17 of 34 97 of 649 words English (United Kingdom) 152%

Type here to search 21°C Mostly cloudy ENG 23:55 02/09/2021

Polymorphism





Source: https://www.youtube.com/watch?v=SiBw7os-_zI