



# Welcome to the CoGrammar Software Design

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.



# Software Engineering Session Housekeeping

---

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.  
**(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

## Software Engineering Session Housekeeping cont.

---

- For all **non-academic questions**, please submit a query:  
[www.hyperiondev.com/support](http://www.hyperiondev.com/support)
- Report a **safeguarding** incident:  
[www.hyperiondev.com/safeguardreporting](http://www.hyperiondev.com/safeguardreporting)
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

# Skills Bootcamp

## 8-Week Progression Overview

### Fulfil 4 Criteria to Graduation

#### ✓ Criterion 1: Initial Requirements

- ***Guided Learning Hours (GLH):***  
Minimum of 15 hours
- ***Task Completion:*** First 4 tasks

**Due Date:** 24 March 2024

#### ✓ Criterion 2: Mid-Course Progress

- ***Guided Learning Hours (GLH):***  
Minimum of 60 hours
- ***Task Completion:*** First 13 tasks

**Due Date:** 28 April 2024

# Skills Bootcamp Progression Overview

## ✓ Criterion 3: Course Progress

- **Completion:** All mandatory tasks, including Build Your Brand and resubmissions by study period end
- **Interview Invitation:** Within 4 weeks post-course
- **Guided Learning Hours:** Minimum of 112 hours by support end date (10.5 hours average, each week)

## ✓ Criterion 4: Demonstrating Employability

- **Final Job or Apprenticeship Outcome:** Document within 12 weeks post-graduation
- **Relevance:** Progression to employment or related opportunity

# Learning Objectives & Outcomes

- Comprehend use case analysis for capturing user requirements, including identifying actors and scenarios
- Create sequence and Class diagrams that display the interactions of objects and their relationships within their programs.
- Apply CRUD Matrices to display entity permissions and operations.
- Build Python programs that utilise the MVC pattern.

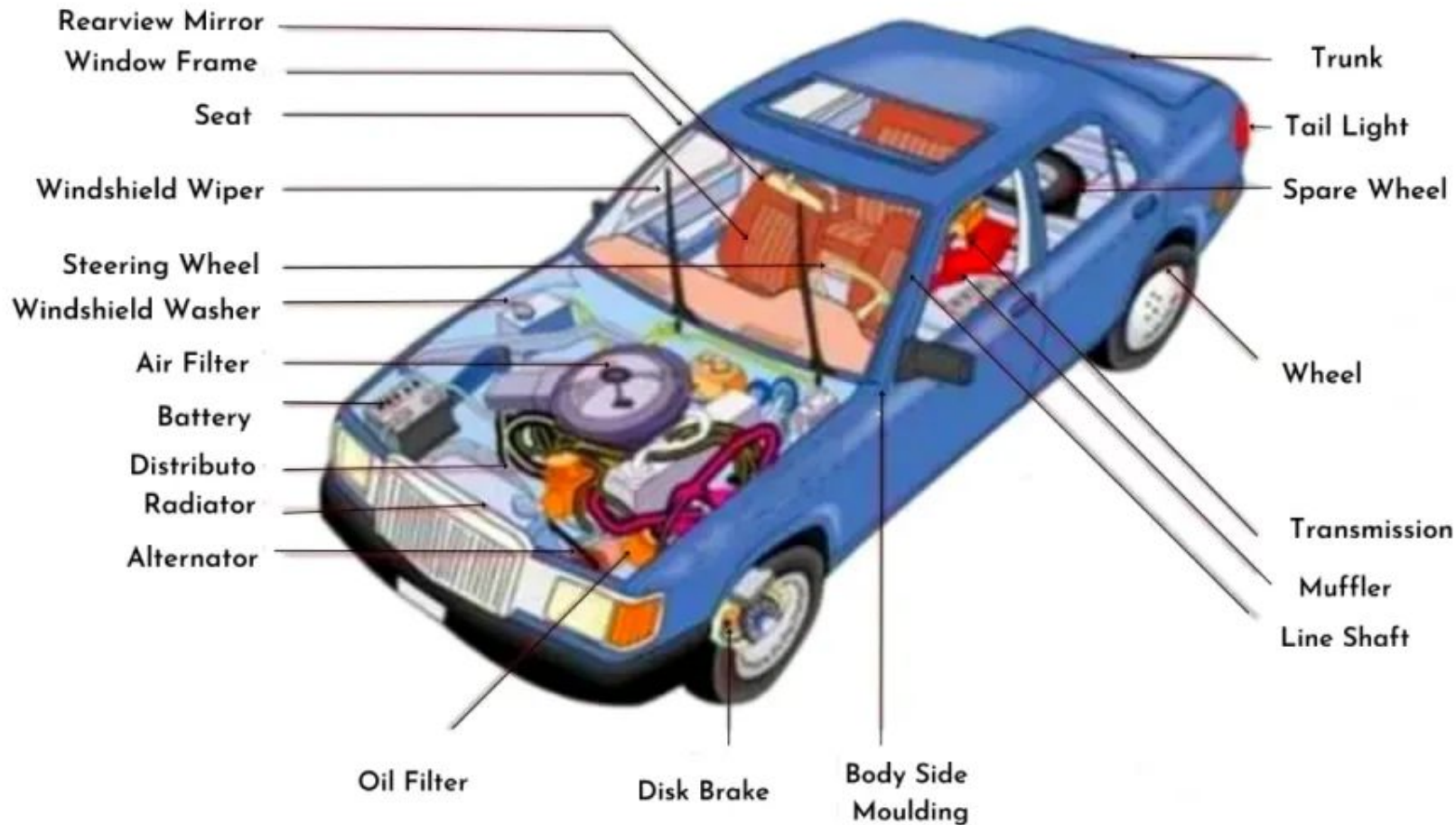


# CoGrammar

## Modular Programming

April 2024

# PARTS OF A CAR





# Definition and Importance

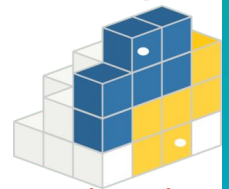
- Modularity in software design is a structured approach that aims to streamline complexity by breaking down systems into distinct, reusable modules.
- These modules encapsulate specific functionality, allowing for easier extension, modification, and integration into various contexts.

# Techniques and Approaches

- This design principle also enhances portability, as modules can be transferred across different environments or platforms with minimal adjustments.
- Modularity promotes maintainability by isolating changes and facilitating debugging and updates, ultimately improving the robustness and flexibility of software systems.

# Benefits of Modularization

- **Collaboration:** Team members can work on different modules concurrently, reducing conflicts and dependencies.
- **Maintainability:** The codebase is easier to understand, update, and debug the code. Changes or fixes can be made to individual modules without affecting other parts of the system, reducing the risk of unintended consequences.
- **Reusability:** Modular programming promotes the creation of independent modules or components that can be reused in different parts of the software system or even in other projects. This reduces duplication of code and saves development time.
- **Ease of Refactoring:** Developers can modify or improve individual modules without affecting other parts of the system, making it safer and more efficient to refactor code to improve its structure, readability, and maintainability.



GitLab



# Benefits of Modularization

- **Testing and Debugging:** Modular programming simplifies testing and debugging efforts as modules can be tested independently of each other. This modular approach allows for more focused and efficient testing, leading to higher code quality and fewer bugs in the final product.
- **Flexibility:** Modular programming enables developers to modify or replace individual modules without impacting other parts of the system. This flexibility allows for easier adaptation to changing requirements, technological advancements, or business needs over time.
- **Scalability:** Modular programming facilitates the scalability of software systems by allowing developers to add new features or functionalities as separate modules. This modular approach makes it easier to extend the system's capabilities without having to overhaul the entire codebase.



pytest

JUnit

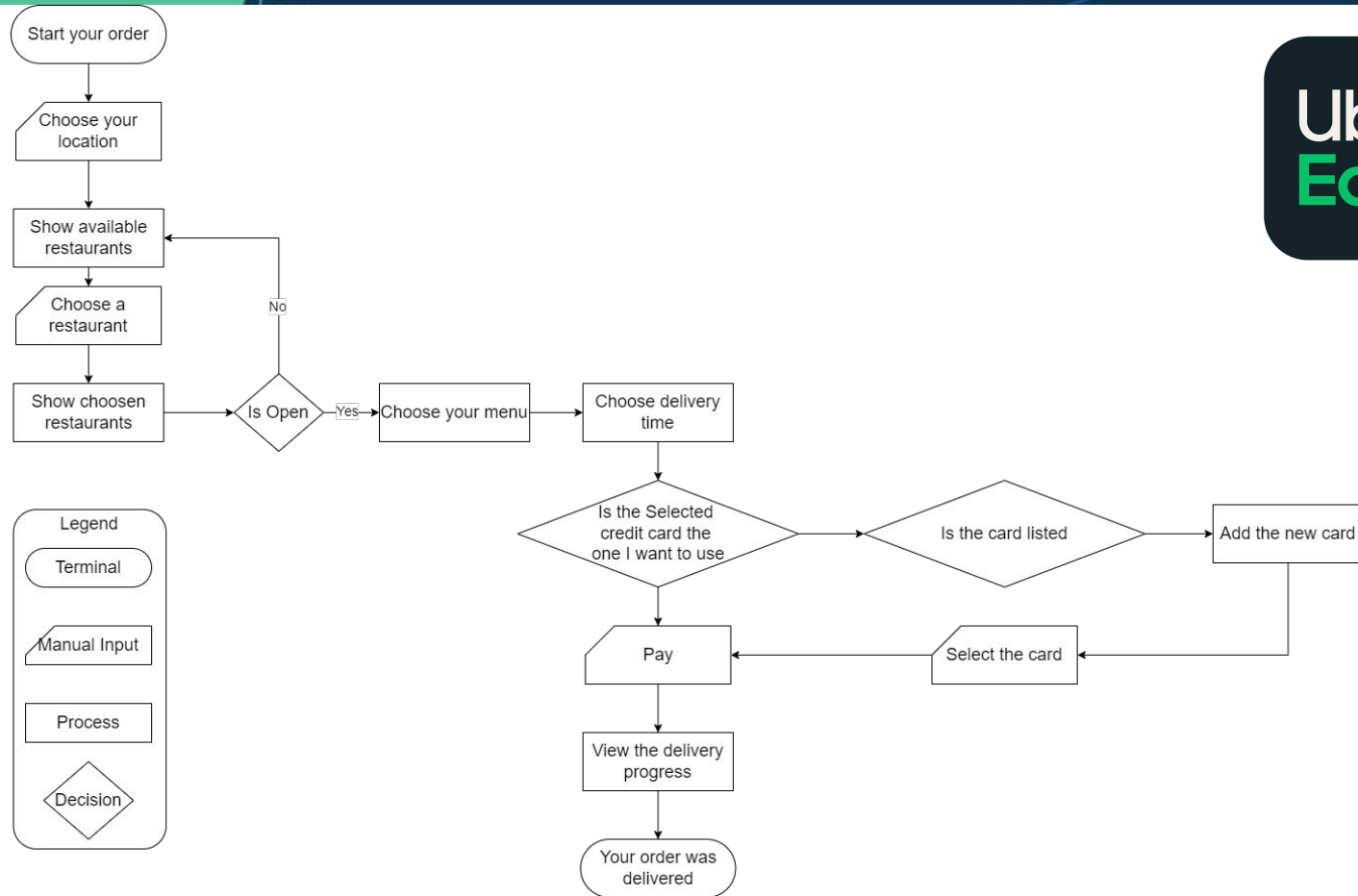


# CoGrammar

## Sequence Diagrams

April 2024







# Intuition






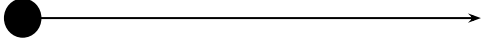
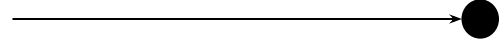
Before diving into the development of a platform like UberEats, it's crucial to map out the sequence of events that occur behind the scenes. Imagine you're craving your favourite meal and open the app to place an order. But what happens next?

How does the app communicate with the restaurant and ensure your food arrives hot and on time? That's where Sequence diagrams come in. They help us visualize the entire journey, from user interaction to backend processing, ensuring every step is carefully orchestrated for a seamless experience.

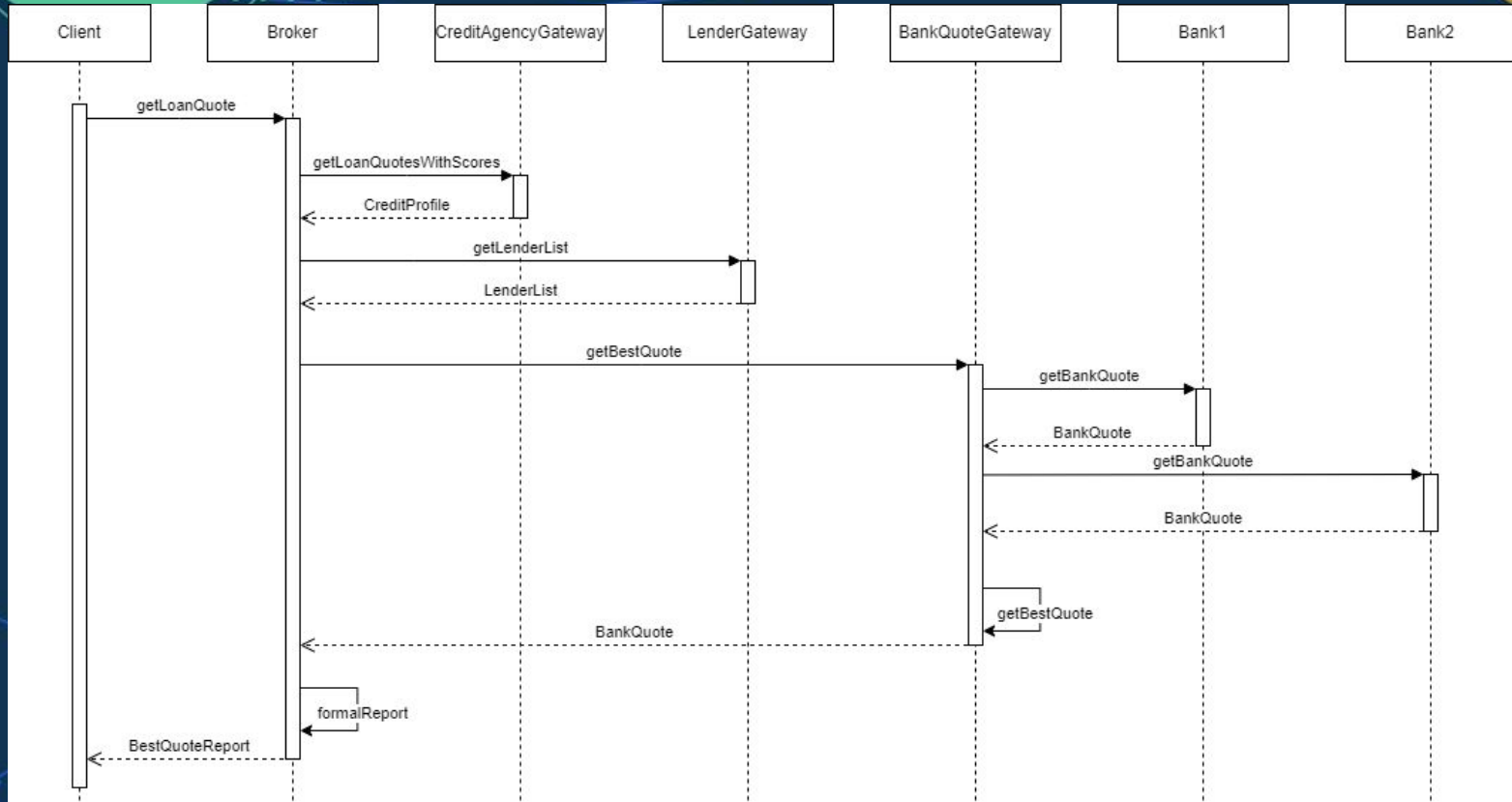
# Sequence Diagrams Basics

- Shows control flow, the order of interactions
- Time runs vertically, from top to bottom
- Messages run horizontally
- Type of UML diagram

# Sequence Diagrams Key Components

Synchronous message	
Asynchronous message	
Message return	
Object creation	
Object destruction	
Found message	
Lost message	

# Use Case



# CoGrammar

## Use Case Analysis

April 2024



# Intuition

Behind the scenes, there's a complex process involving various stakeholders and systems. Use case diagrams provide a bird's-eye view of this process, capturing the different interactions between actors (such as customers, loan officers, and administrators) and the system itself.

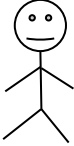

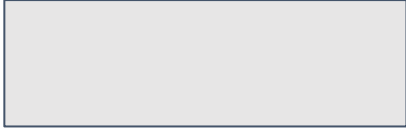
By visualizing the different scenarios and functionalities required to facilitate loan processing, use case diagrams help us understand the core functionalities of how different actors interact with it to achieve their goals efficiently.




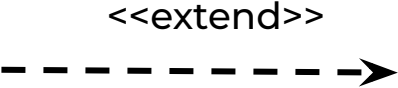



# Use Case Diagrams Basics

- Describe functionality from the user's perspective
- One (or more) use-cases per kind of user
  - May be many kinds in a complex system
- Use-cases capture requirements
- Type of UML diagram

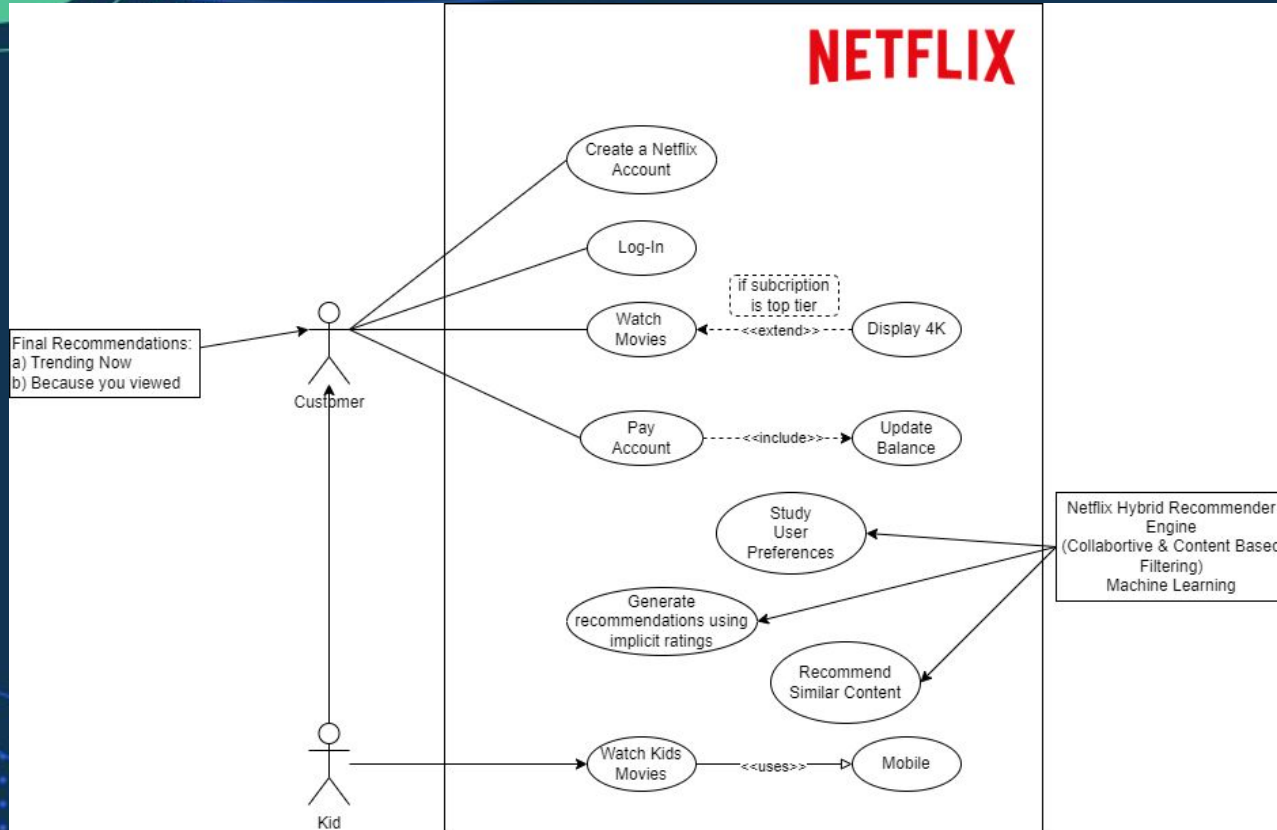
# Use Case Diagrams Key Components

Actor		Human or system interacting with the system. People, Devices, External Systems
Use Cases		
System		Helps identify what is external versus internal, and what the responsibilities of the system are.

# Use Case Diagrams Key Components

Association		A actor must be associated with at least one use case Multiple actors can be associated with one use case
Extend		To extend the functionality of a use case, given a condition.
Generalisation		An actor can inherit the role of another one
Uses		When a use case uses another process
Include		SHow the included or implicit behaviour of a use case

# Syntax and semantics



# CoGrammar

## Separation of Concerns and MVC pattern

April 2024



# Separation of Concerns (SoC)

- SoC is a design principle that advocates breaking a program into distinct sections, each addressing a separate concern or aspect of functionality.
- By separating concerns, such as data management, user interface, and business logic, developers can focus on one aspect at a time, leading to cleaner, more modular, and easier-to-maintain code.



# Benefits of SoCs

- **Modularity:** SoC encourages the creation of modular components, each responsible for a single aspect of the system's behavior.
- **Code Reusability:** By separating concerns, developers can create reusable components that can be leveraged across multiple parts of the system or in different projects.
- **Maintainability:** SoC improves the maintainability of software by making it easier to locate and fix bugs, add new features, or make changes without affecting other parts of the system.

# Benefits of SoCs ...

- **Scalability:** As the system grows, new components can be added or existing components can be modified or replaced without disrupting the entire system architecture.
- **Concurrent Development:** SoC enables multiple developers to work on different modules simultaneously.
- **Testing and Debugging:** As each component can be tested independently of the rest of the system, this isolation reduces the scope of testing and makes it easier to identify and fix issues.

# Model-View-Controller Pattern

- MVC is a software architectural pattern commonly used in web and application development.
- MVC divides an application into three interconnected components:
  - Model represents the application's data and **business logic**, independent of the user interface.
  - View displays the data to the user and handles user interactions, such as input and output and therefore represent the **presentation layer**.
  - Controller **processes user input**, interacts with the model to update data, and updates the view accordingly.

# Benefits of MVC Pattern

- MVC promotes separation of concerns by keeping the data (Controller), user interface (View), and application logic (Model) separate, making the codebase more organised and maintainable.
- This separation allows developers to make changes to one component without affecting others, facilitating parallel development and easier maintenance.
- MVC promotes code reusability, as the same model can be used with different views or controllers, enhancing the flexibility and scalability of the application.

**SKILLS  
FOR LIFE**

**SKILLS BOOTCAMPS**



Department  
for Education

# CoGrammar

## Class Diagrams

April 2024



# Class Diagrams

- Class diagrams are a type of Unified Modelling Language (UML) diagram used to visualise the structure and relationships of classes in an object-oriented system.
- They provide a high-level overview of the relationships between classes, attributes, and methods in an object-oriented system.



# Class Diagrams ...

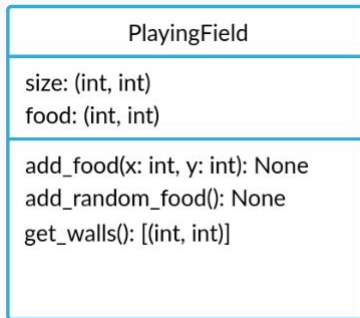
- Class diagrams help developers understand the architecture of a system, identify relationships between classes, and communicate design decisions with stakeholders.
- They serve as blueprints for software development, also aiding in identifying potential design flaws or bottlenecks early in the development process, allowing for timely adjustments and optimizations.

# Key Elements: Class

- Classes encapsulate data and behaviour, defining the blueprint for creating objects in the system.
- In the diagram, a class is represented as a rectangle with three compartments. The top compartment contains the class name, the middle compartment lists the class attributes, and the bottom compartment shows the class methods.

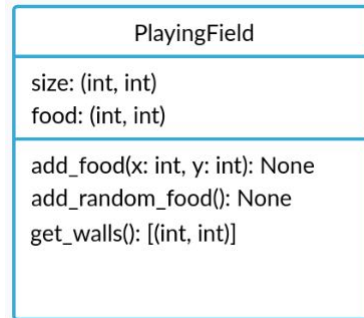
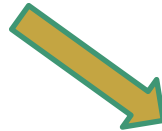
# Key Elements: Attributes

- Attributes represent the data or properties associated with a class. They describe the state of objects belonging to that class.
- In the diagram, Attributes are listed in the middle compartment of the class rectangle, usually preceded by their visibility (e.g., public (+), private (-) and followed by their data type.



# Key Elements: Methods

- Methods describe the actions or functionalities that objects of the class can exhibit.
- In the diagram, Methods are typically listed in the bottom compartment of the class rectangle, beneath the attributes and usually preceded by their visibility (e.g., public (+), private (-) ).

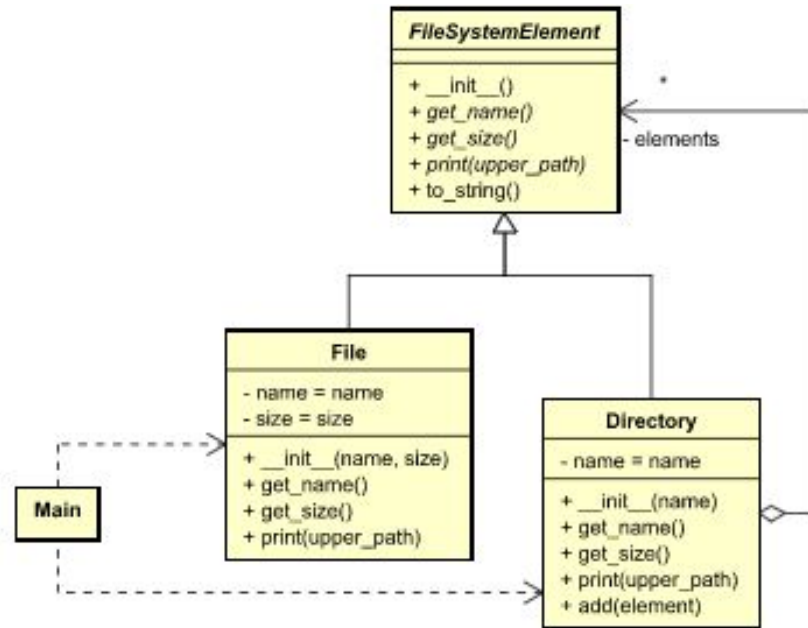


# Key Elements: Relationships

- Relationships represent the associations and dependencies between classes in the system.
- They describe how classes interact with each other and can be categorised into several types.



# Class Diagram: Example



**SKILLS  
FOR LIFE**

**SKILLS BOOTCAMPS**



Department  
for Education

# CoGrammar

## CRUD Matrices

April 2024

# CRUD Matrices

- CRUD stands for **Create**, **Read**, **Update**, and **Delete**, which represents the fundamental operations performed on data within a software system.
- CRUD Matrices serve as a roadmap for organizing and documenting the functionalities of classes, helping developers ensure that their software systems can efficiently manage data.

# CRUD Matrices: Example

- Basic CRUD Matrix

Entity	Create	Read	Update	Delete
Book	X	X	X	X
Author	X	X	X	X
Publisher	X	X	X	X
Customer	X	X	X	X
Order	X	X	X	X
Review	X	X	X	X

# CRUD Matrices: Example

- Role-Based CRUD Matrix

Entity	Admin	Manager	Customer
Book	CRUD	CRU	R
Author	CRUD	CRU	R
Publisher	CRUD	CRU	R
Customer	CRUD	CRU	R
Order	CRUD	CRU	R
Review	CRUD	CRU	R



# CRUD Matrices

- It maps out **which users or roles have access to perform CRUD operations** on specific data entities.
- CRUD matrices help ensure data integrity, security, and compliance with access control requirements by clearly defining who can perform which actions on the data.
- CRUD matrices also facilitate compliance with regulatory requirements and industry standards by documenting data access policies and audit trails.

**Let's take a short  
break**

**CoGrammar**



# Summary

- **Modularization:** Breaking down software into independent modules enhances maintainability, scalability, and reusability.
- **Sequence Diagrams:** Visualising component interactions elucidates system behaviour over time, aiding in comprehension and optimization.
- **Use Case Diagrams:** Representing system functionalities from user viewpoints aids in requirement analysis and stakeholder communication.
- **Separation of Concerns:** Emphasizes dividing a software system into distinct sections, each responsible for a separate concern or aspect of functionality.
- **MVC:** software architectural pattern widely used in web development. It divides an application into three interconnected components: Model (data management), View (user interface), and Controller (business logic).
- **Class diagrams:** are tools used in software development to plan and organize class functionalities.
- **CRUD matrices:** tools used in software development to plan and organize class functionalities.

# Questions and Answers



# Thank you for attending



Department  
for Education

CoGrammar

