Welcome to this CoGrammar session:

Class Diagrams and CRUD Matrices

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**
- Report a **safeguarding** incident: **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

CoGrammar

Learning Objectives & Outcomes

- Understand the concept of **Separation of Concerns** and its importance in software engineering.
- Describe the **Model-View-Controller** (MVC) **pattern** and its role in software development.
- Create and interpret **Class Diagrams** to visualise the structure of software systems.
- Explain the purpose of **CRUD Matrices** in software development and use them to design database systems.

Polls

- *Refer to the polls section to vote for you option.*

1. What is Separation of Concerns in software engineering?

a. It refers to the division of a program into distinct sections for different functionalities.

b. It means focusing on only one aspect of a problem at a time.

c. It involves separating HTML, CSS, and JavaScript in web development.

d. It's a design pattern used to organise code into Model, View, and Controller components.

Polls

- *Refer to the polls section to vote for you option.*

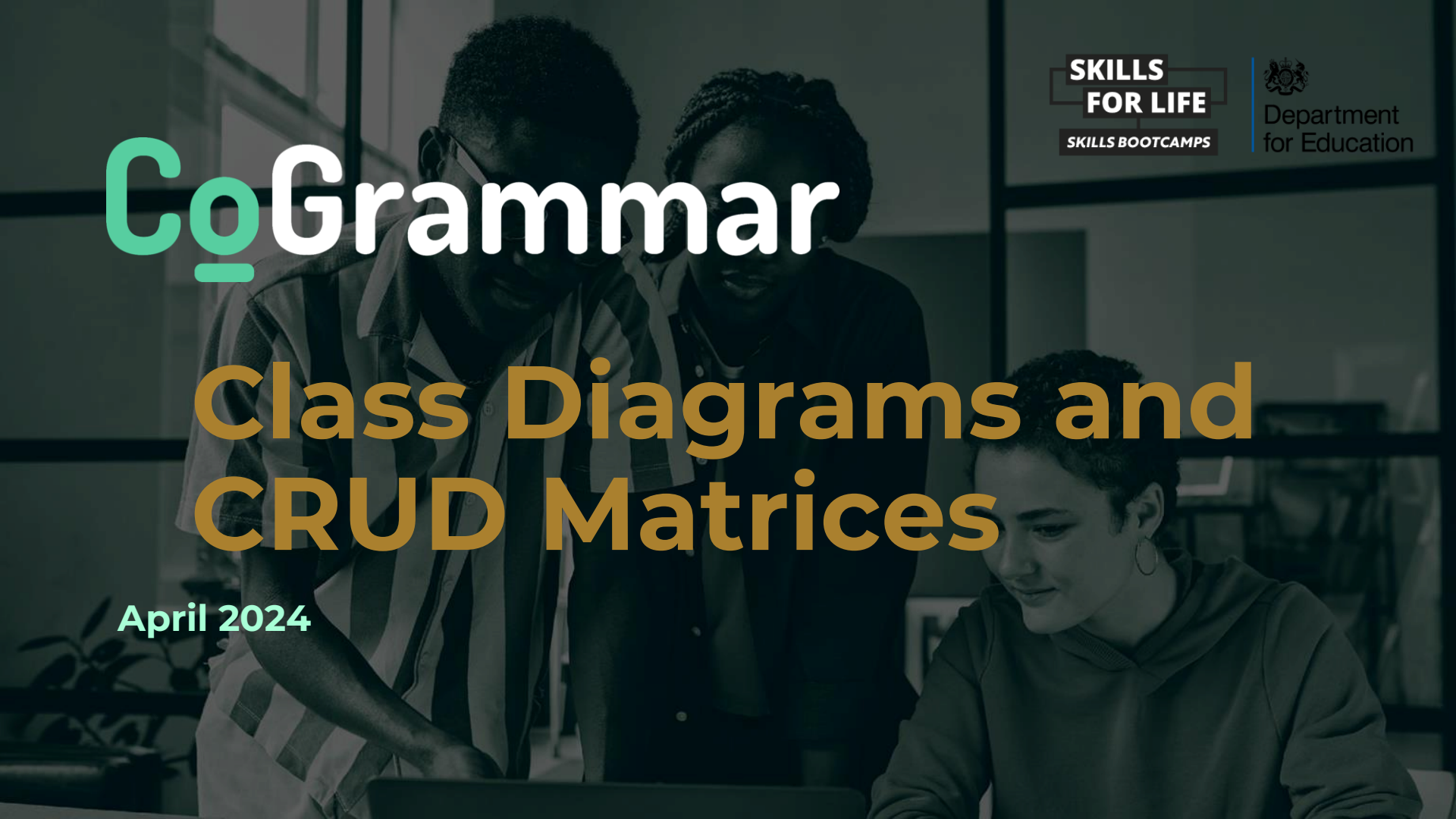
2. What is the primary purpose of a Class Diagram?

a. To visualise the flow of control in a program.

b. To depict the interactions between database tables.

c. To represent the structure and relationships of classes in an object-oriented system.

d. To illustrate the sequence of operations in an algorithm.



CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Class Diagrams and CRUD Matrices

April 2024

Introduction

- Organising software systems is crucial for better maintainability and scalability, because of:
  - Maintainability
  - Scalability
  - Collaboration and Teamwork
  - Reduced Technical Debt

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 behaviour.
- **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**.

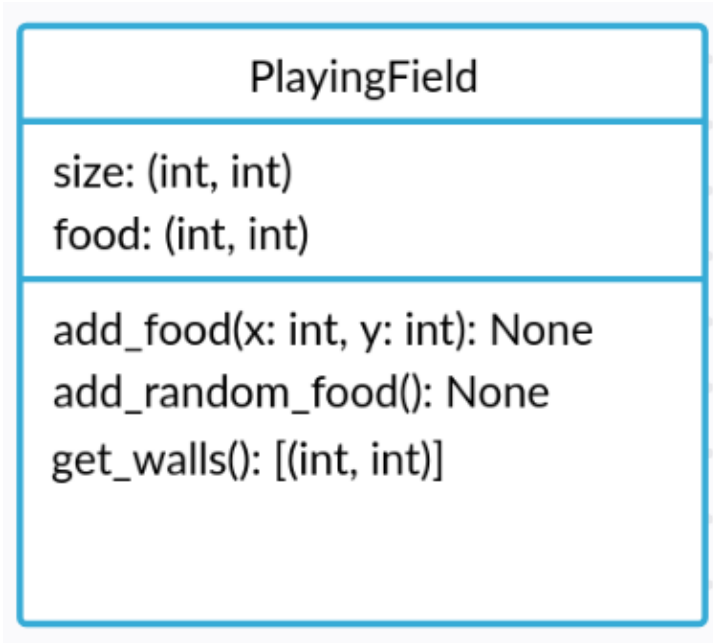
Class Diagrams

- 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 optimisations.

Key Elements: Class

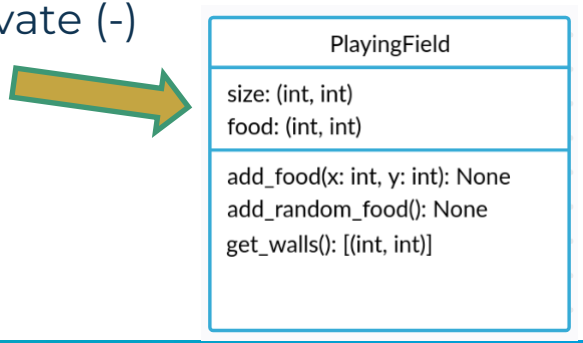


Key Elements: Class

- 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.

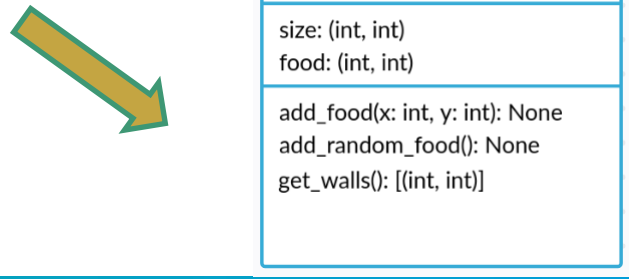
Key Elements: Attributes

- 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 (-) .

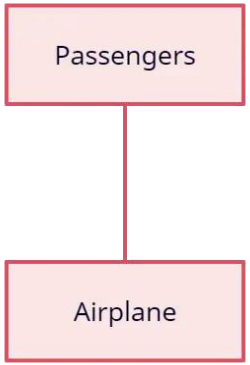


Key Elements: Relationships

- 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.

Relationship: Association

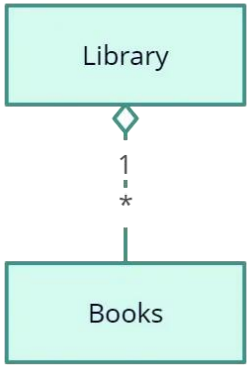
- Represents a relationship between two classes where **one class is related to another**. It can be one-to-one, one-to-many, or many-to-many.



Association

Relationship: Aggregation

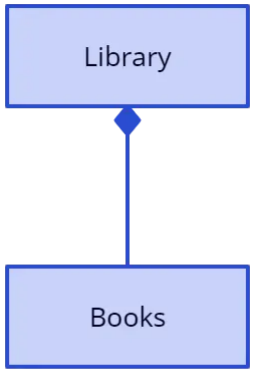
- Represents **a "whole-part" relationship between classes**, where one class (the whole) contains or owns instances of another class (the part).
- It is denoted by a hollow diamond at the containing end.



Aggregation

Relationship: Composition

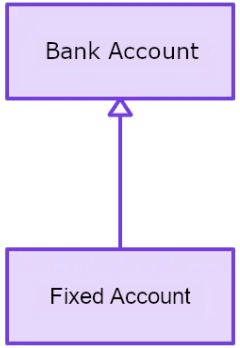
- Similar to aggregation but with stronger ownership semantics. It implies that the **part cannot exist without the whole**.
- It is denoted by a filled diamond at the containing end.



Composition

Relationship: Inheritance

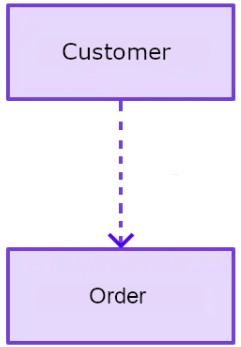
- Represents **an "is-a" relationship between classes**, where one class (the subclass or child) inherits attributes and methods from another class (the superclass or parent).
- It is denoted by a solid line with a hollow arrowhead pointing to the superclass.



Inheritance

Relationship: Dependency

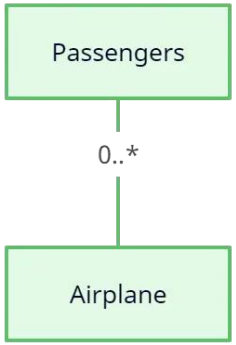
- Represents **a weaker form of relationship** where one class depends on another class.
- It is denoted by a dashed line with an arrow pointing from the dependent class to the independent class.



Dependency

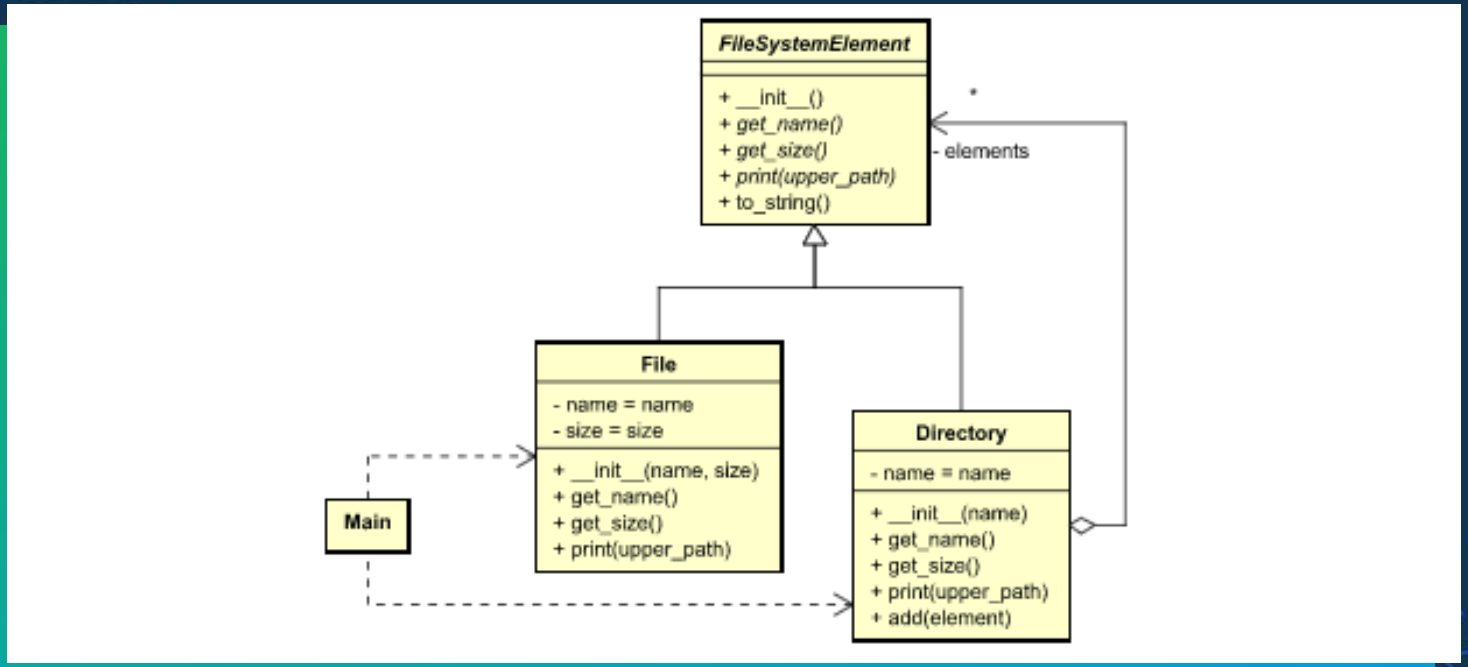
Key Elements: Multiplicity

- Multiplicity specifies the cardinality of associations between classes, indicating **how many instances of one class are related to instances of another class**.
- It is represented using numbers or symbols near the association lines,
  such as "1" for one instance,
  "*" for many instances,
  "0..1" for zero or one instance, etc.



Multiplicity

Class Diagram: Example



CRUD Matrices

- CRUD stands for **C**reate, **R**ead, **U**pdate, and **D**elete, which represents the fundamental operations performed on data within a software system.
- CRUD Matrices serve as **a roadmap for organising and documenting the functionalities of classes**, helping developers ensure that their software systems can efficiently manage data.

CRUD Matrices: Create

- In the context of classes, the "Create" operation refers to the **ability to instantiate or create new objects** of a particular class.
- It **involves** defining **methods** or functionalities within the class that allow users **to create new instances** and initialise their attributes.

CRUD Matrices: Read

- The "Read" operation pertains to accessing or **retrieving data from existing objects** or instances of a class.
- It **includes** defining **methods** or functionalities that enable users **to retrieve specific information** or query data stored within class instances.

CRUD Matrices: Update

- "Update" involves modifying or **updating the attributes** or properties **of existing class instances**.
- It **encompasses methods** or functionalities designed to allow users **to change the values of certain attributes** within objects.

CRUD Matrices: Delete

- Finally, the "Delete" operation revolves around **removing** or deleting **existing** instances or **objects of a class**.
- It **comprises methods** or functionalities that enable users **to delete specific data** or instances from the system.

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

Let’s get coding!

CoGrammar

Polls

- *Refer to the polls section to vote for you option.*

1. **What is a primary benefit of Separation of Concerns in software engineering?**

   a. Improved performance of the system.

   b. Increased modularity and easier maintenance.

   c. Reduction in development time.

   d. Better user interface design.

Polls

- *Refer to the polls section to vote for you option.*

2. In the Model-View-Controller (MVC) pattern, which component is responsible for handling user interactions?

a. Model

b. View

c. Controller

d. Database

Polls

- *Refer to the polls section to vote for you option.*

3. What is the main purpose of a Class Diagram in software engineering?

a. To visualise the flow of control in a program.

b. To represent the structure and relationships of classes in an object-oriented system.

c. To define the database schema.

d. To illustrate the sequence of operations in an algorithm.

Polls

- *Refer to the polls section to vote for you option.*

4. Which CRUD operation involves retrieving data from a class instance?

a. Create

b. Read

c. Update

d. Delete

Polls

- *Refer to the polls section to vote for you option.*

5. How are CRUD Matrices used in software development?

a. To define class attributes.

b. To organise project documentation.

c. To plan class inheritance.

d. To optimise code structure.

Questions and Answers

CoGrammar

Summary

- In today's lesson, we covered several key concepts essential to software design and development.
- Organising software systems enhances **maintainability**, **scalability**, **collaboration**, and **reduces technical debt**.
- It lays the foundation for building robust, adaptable, and sustainable software solutions that can evolve with changing requirements and user needs.

Summary

- **Separation of Concerns** emphasises dividing a software system into distinct sections, each responsible for a separate concern or aspect of functionality.
- By separating concerns, we enhance maintainability, scalability, and code reusability.

Summary

- **MVC** is a 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).
- MVC promotes code organisation, modularity, and separation of concerns.

Summary

- **Class diagrams** are graphical representations of the structure and relationships of classes within a system.
- Class diagrams provide a visual overview of the system's architecture, aiding in design, communication, and documentation.

Summary

- **CRUD matrices** are tools used in software development to plan and organise class functionalities.
- CRUD matrices facilitate clear understanding and implementation of class functionalities.

Homework

- **Library Management System:**
  - Implement features such as adding new books (Create operation), searching for books (Read operation), updating book details (Update operation), and deleting books (Delete operation) within the MVC framework.

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar