brunkonjaa

Object-Oriented Software Engineering (OOSE)

Week 1 - 2

**UML Basics**

* **UML = Unified Modelling Language**
  + - a standard way to draw system blueprints.
* Two main groups of diagrams:
  + - **Structural diagrams** → what the system *is* (nouns).
    - **Behavioural diagrams** → what the system *does* (verbs).

**Why it works like this:**

* Systems are complex. Splitting diagrams into *structure* vs *behaviour* makes them easier to understand.
* Like anatomy vs movement: skeleton = structure,  
  actions = behaviour.

**Use Cases**

**What is a Use Case?**

* A short story of how an **actor** interacts with a **system** to achieve a **goal**.
* Drawn as ovals inside the system boundary.

**Actors**

* **Primary Actor (PA):** whose goal drives the use case.
* **Supporting Actor (SA):** helps make it happen.
* One person/system can be PA in one case, SA in another — but never both in the same case.

**System Boundary**

* A box that separates the **system** (inside) from the **actors** (outside).

**Relationships**

* **Association:** line linking actor ↔ use case.
* **Include:** one use case always uses another.
* **Extend:** one use case sometimes adds to another.
* **Generalisation:** specialisation (actors or use cases).

**Steps to Build a Use Case Diagram**

1. Draw the system boundary (box).
2. Place actors outside.
3. Place use cases (ovals) inside.
4. Connect actors ↔ use cases.
5. Add include/extend/generalisation if needed.

**Why it works like this:**

* Instead of a wall of text, a use case diagram shows **who does what** with the system at a glance.
* It highlights goals and roles without drowning in code or detail.

**Object-Oriented Principles**

1. **Abstraction**

* Focus on **what something does**, not **how it does it**.
* Example: A “Payment” shows amount and date but doesn’t reveal if it’s cash or card.

**Why it works like this:**

* Details make systems heavy.
* Abstraction strips noise away so design is clearer and easier to change.

1. **Inheritance**

* A class can reuse attributes and methods from another.
* Example: Property class holds rooms + location. Flats, Houses, Commercial inherit these and add unique features.

**Why it works like this:**

* Avoids code repetition.
* Centralises shared details so one change updates everything.

1. **Encapsulation**

* Data + methods bundled into one object.
* Object hides how it works; only exposes what’s needed.
* Example: Asking for a “green ball” — you don’t see the process, just the result.

**Why it works like this:**

* Prevents fragile code where everyone can poke the data.
* Limits the effect of changes to one safe place.

1. **Polymorphism**

* “Many forms” — one interface, many behaviours.
* Example: Animals all “speak,” but each makes a different sound.
* Analogy: sockets and bulbs — one socket, many bulb types fit.

**Why it works like this:**

* Without polymorphism, every type needs a special connection.
* Interfaces let us swap components freely → flexible, future-proof design.

**OOSE Conclusion**

* **UML** → shows structure and behaviour visually.
* **Use Cases** → tell stories of how actors use a system to reach goals.
* **Principles (A, I, E, P)** → keep systems simple, reusable, and adaptable.

Together, these give you the tools to design **software structure and behaviour** before a single line of code is written.