Modelling Classes

Object Modeling

- Very fundamental.
- Drawing a diagram or writing some code is object modeling.
- Modeling is a representations or abstraction or idea. We represent this in our code.

What is it?

- It is a way to cope up with the complexity of what we are building, so we can communicate those effectively.

Static Structure

- An object model is a static structure.
- A sequence diagram shows the flow of a particular function through different objects and methods. Very helpful in complex situations. In most cases, it is overkill.
- An interaction diagram is different from a sequence diagram in that, we show the interaction between object by which objects collaborate with other objects and not through the timeline.

Objects, Relationships

- An object diagram and a class diagram represent how objects and classes are placed, how they interact, their relationships, and what they contain.
- The diagram should be very light weight. Should contain only the essential characteristics. No unnecessary detail.

What to show?

- Show only the most important detail in a class diagram. Class diagram is not a full representation of a system, but the most essential aspect that you would like to

show.

Why use it?

Used in two phases →

1. During the initial design phase. A strategic design. Represent your design idea. To talk about pros and cons. To evolve the design. For brainstorming purposes.
2. Using as communication tool. If there's a new member or someone who wants/needs to know how things are working.
    a. Serves as a nice way to description the relationship between classes and how they interact with each other.

Objects

What's an object?

- Something that represents an abstraction. It stands in for something in the application.
- Could be tangible or intangible.
    - Example of tangible: representing a cup or a coaster.
    - Example of intangible: representing a thought or an idea, or a message passing through a message queue.

Object Decomposition

- Objects have identities. We can differentiate between objects.
    - Eg. this person is different from that person.
    - Domain specific identifiers are used to distinguish one object from another.
        - Eg. in Databases, often, the primary key is the identifier.
        - Eg. in a running program, the location in the main memory could be the identifier.

- When we do Object Decomposition, we take a problem on hand and assign responsibilities to different objects.
    - “This object is responsible for this kind of behaviour and state. This other object is responsible for another behaviour and state.”
    - Eg. The `Presentation` object may be responsible for keeping track of presentation details, such as the title of the presentation, the file of the presentation, the length, who the presenter is. The `Presenter` is another object itself. The `Presentation` and the `Presenter` objects have a relationship, but they have their own attributes/properties.
- When it comes to decomposing the application, we can usually based on the domain knowledge and the usage patterns.
    - Eg. The presentation, presenter, user are all objects. But the duration of the presentation is not an object by itself — it is an attribute.
    - We have to look at the usage pattern and ask — “Does this deserve to have its own identity?”
        - Eg. presentation deserves its own identity, but the duration of the presentation does not.

No single correct representation

- Pick and choose based on the context and the bounded context.

Classes

- Often a blueprint of objects. Eg. a collection of objects become a class.

How to recognise them?

- A class should have a very clear representation of an abstraction. It’s got a very specific meaning in your application.
- Class often have state, but they also have behaviour. So you could capture the state and behaviour into a class and that’s how you represent it in your application.

- Classes often/typically show up as nouns. However, be careful and look at the context — a noun could be an attribute of a class.
    - When creating classes, try to name them as nouns so you know that they represent something.
    - Usually methods are verb forms.

Semantics, Abstraction

- There are semantics and abstraction, which we have to define based on the context, etc.

Interpretation Varies

- Depends on the application.
- Depends on the person modelling.
    - The individual modelling will learn new things — so will design in a different way a year from now, given the same prompt.

Instances

- Typically, a class can have as many instances as you want → 0 to thousands.

Singletons

- Class that has only one (or even a very small set of) object at any given time.

Abstract Classes

- Class that has zero direct object.
- Used to model a concept that gets implemented and extended.

Class diagram vs. Object diagram

- A class diagram is one that shows classes and the interactions between them.

- An object diagrams shows objects and the interaction between them.
- Often, we create class diagram, but occasionally object diagrams are also made.
    - Interaction diagram are often an object diagram.

Object Diagram

- Objects and Relationships.
- Useful to document test cases.
- Clarify complex behaviours or interactions.
- Do not create these unless you need to explain some intricate relationship.

Class Diagram

- Used most of the time, to create static relationship between classes.
- Should be concise, easy to understand, practical.
    - The more a class diagram has, often, the less effective it is.
- Keep it lightweight

Representing a class

- UML — Unified Modeling Language. Used to communicate design ideas.

Attributes

- Data value that you have within a class.
- Attributes may be different for different instances.
- Name is unique within a class.
- Adjectives often as enumerate attribute values.
    - Eg. what’s the size of a shirt (S, M, L, XL, etc).
    - Eg. Red car, Blue car, etc.

- Represent pure data values as attributes. In C++, we call these member variables. In UML, we call them attributes. In Java, C#, they are called fields.
- Represent object values as relationships.
  - Eg. a `Person` class won't hold a `Car` object in its attributes if `Car` has a separate identity—have a relationship.
- Don't show internal identifiers.
- Don't show every single attribute.
- Visibility. In UML →
  - Use a minus − for a private attribute.
  - Use a plus + for a public attribute.
  - Use an octothorp # for protected.
  - However, these notations in UML are different for different languages.

Derived Attributes

- Base attribute → an attribute that is very primary. Eg. date of birth becomes a primary attribute.
- Computed value → a derived attribute. Eg. how old is a person? Age is not being age, so calculate using dob.

Operations and Methods

- Operations → used in C++ often. In Java/C#, method term is used. It is something that you can do on an object.
- A method is often an implementation of an operation on an object.
- Polymorphic methods → a method that is available on multiple objects, but the implementation on different objects varies.
- Query operations → `const` methods in C++. Methods that don't do any mutations. No such concept in Java/C#.
  - Eg. `getAge()`, the state does not change, it's just a query method.

- Mutators → methods that often modify the state of an object. Eg. `run()` would change the location of the person.
- Command query separation → where a method which is a command on an object, and a query is where you ask for a detail.