OBJECT ORIENTED ANALYSIS & DESIGN DATA STRUCTURES & ALGORITHMS

Object and Object Behavior

Object Oriented Philosophy

- Object-Orientation is what’s referred to as a programming paradigm. It’s not a language itself but a set of concepts that is supported by many languages.
- It’s a structured method for analyzing, designing a system by applying the object-orientated concepts, and develop a set of graphical system models during the development life cycle of the software.

Objects and Object Behavior

- We are oriented or focused around objects.
- Now in an object-oriented language, this one large program will instead be split apart into self contained objects, almost like having several mini-programs, each object representing a different part of the application.
- And each object contains its own data and its own logic, and they communicate between themselves.
- These objects aren’t random. They represent the way you talk and think about the problem you are trying to solve in your real life.
- They represent things like employees, images, bank accounts, spaceships, asteroids, video segment, audio files, or whatever exists in your program.

Object Oriented Properties

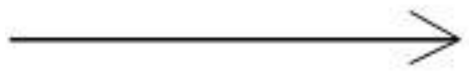
In the object-oriented analysis, we…

Elicit requirements: Define what does the software need to do, and what's the problem the software trying to solve.

Specify requirements: Describe the requirements, usually, using use cases (and scenarios) or user stories.

Conceptual model: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

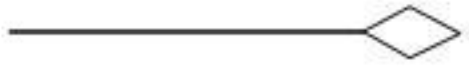
Association



Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.

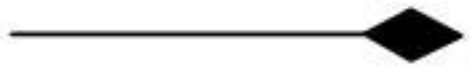
Example: A Student and a Faculty are having an association.

Aggregation



Aggregation is a special case of association. A directional association between objects. When an object 'has-a' another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a “Has-a” relationship.

Composition



Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

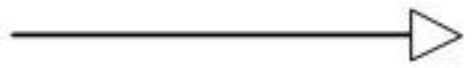
Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

Abstraction

Abstraction is specifying the framework and hiding the implementation level information. Concreteness will be built on top of the abstraction. It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.

Example: A wire frame model of a car.

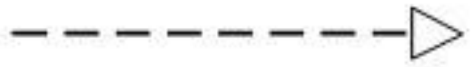
Generalization



Generalization uses a “is-a” relationship from a specialization to the generalization class. Common structure and behavior are used from the specialization to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a “Is-a” relationship.

Example: Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

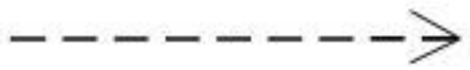
Realization



Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

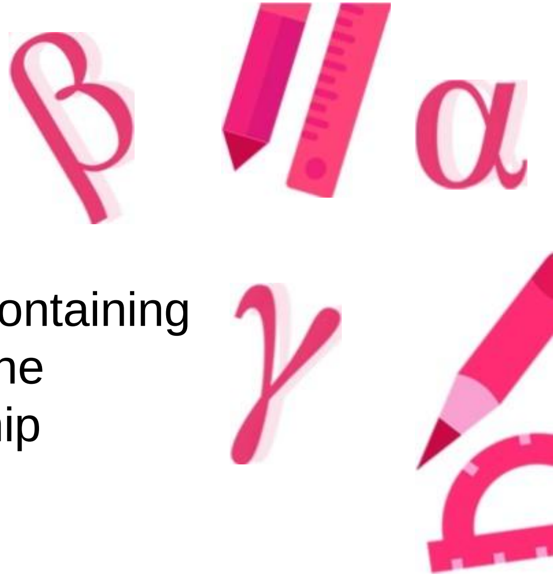
Example: A particular model of a car ‘GTB Fiorano’ that implements the blueprint of a car realizes the abstraction.

Dependency



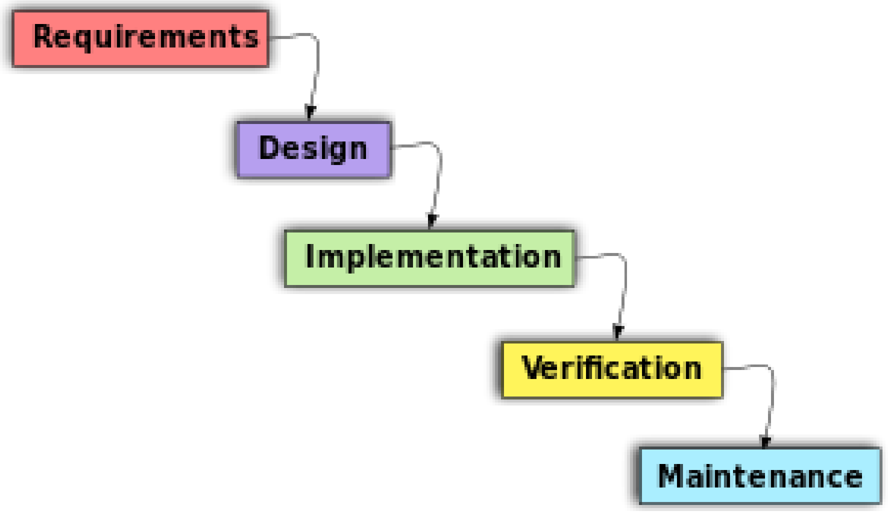
Change in structure or behavior of a class affects the other related class, then there is a dependency between those two classes. It need not be the same vice-versa. When one class contains the other class it this happens.

Example: Relationship between shape and circle is dependency.



The Process of Software Development

The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment.



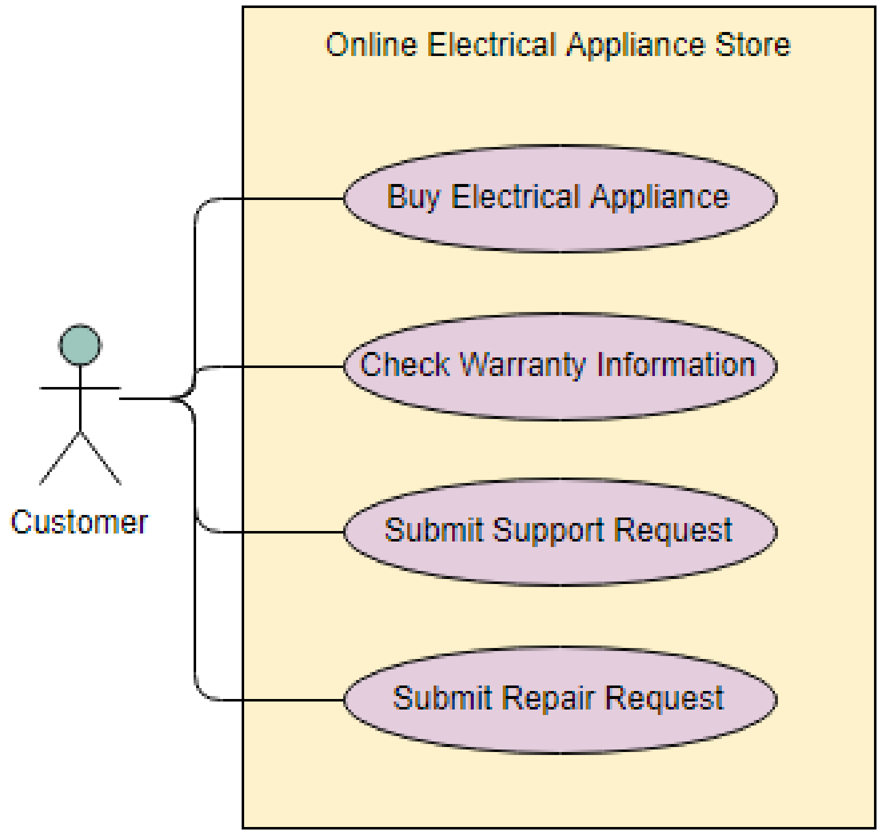
Developing Good Quality Software

- Use code coverage analysis to measure testing completeness
- Improve test coverage with unit tests
- Make tests easy to run, and test results easy to understand
- Implement automated, parallel, and change based testing
- Constantly re-factor code to improve maintainability

Use Case Driven Approach for Object Oriented Systems Development

- Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases.
- A use case describes how a user uses a system to accomplish a particular goal. A use case diagram consists of the system, the related use cases and actors and relates these to each other to visualize: what is being described? (system), who is using the system? (actors) and what do the actors want to achieve? (use cases)
- Thus, use cases help ensure that the correct system is developed by capturing the requirements from the user's point of view.

Use Case Driven Approach for Object Oriented Systems Development



REUSABILITY

- Reusability is the use of existing assets in some form within the software product development process.

- ssets are products and by products of the software development life cycle and include code, software components, test suites, designs and documentation

Thank You!