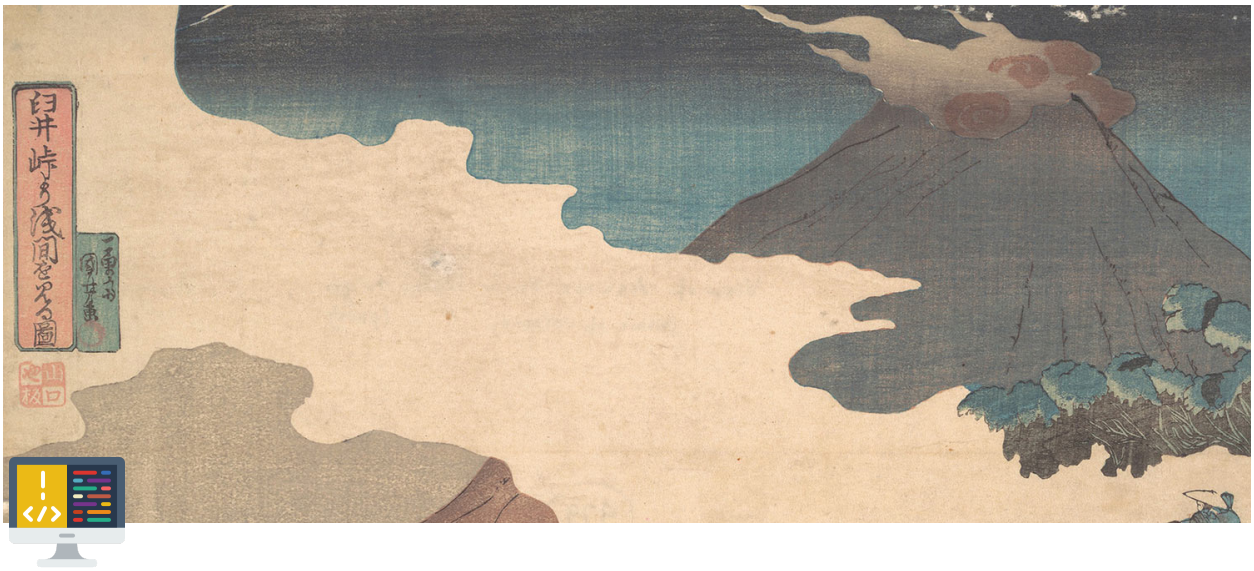


System/Requirements Analysis

Chapter 1 of UML Classroom textbook notes:

Introduction to Object Orientation:

Model proposed by Herbert Stachowiak:

1. *Reduction:* This property is defined as the choice of what aspects of reality to use in the model by reducing it for it purposes.
2. *Mapping:* This property is defined as the image of something - the abstraction of the future action, basically the predication of the action.
3. *Pragmatism:* pragmatism means orientation toward functionality and usefulness.

Quality of the model:

- *Abstraction:* the model is a reduced view of the reality it represents, some details are irrelevant and should be removed for the user understanding of it´s essence.
- *Understandability:* the property of understandability is defined as the representation of complex concepts on an easy manner, this reduces the intellectual effort for the

user or reader to understand the model.

- *Accuracy:* the model needs to feed the system requirements and be the closer of the reality as possible.
- *Predictiveness:* the model needs to be able to predict via analysis or simulation interesting but not obvious properties of the modeled system.
- *Cost-effectiveness:* in the long term it must be cheaper to create the model than to create the system being modeled.

Object orientation:

The object oriented notation used in the UML is defined as the view which humans see the world. The agents like society, individuals and etc... are viewed as objects which have fixed places and obligations.

> There is not only one single definition for object orientation. However,
> there is a general consensus about the properties that characterize object orientation. Naturally, objects play a central role in objectoriented
> approaches. Viewed simply, objects are elements in a system whose data and operations are described. Objects interact and communicate
> with one another.

Classes:

Fundamentally classes define the behavior of objects, meaning that they are the attributes and their common features.

> In many object-oriented approaches, it is possible to define classes that
> describe the attributes and the behavior of a set of objects (the

> instances
> of a class) abstractly and thus group common features of objects. For
> example, people have a name, an address, and a social security number.
> Courses have a unique identifier, a title, and a description. Lecture halls
> have a name as well as a location, etc. A class also defines a set of permitted operations that can be applied to the instances of the class.
> For example, you can reserve a lecture hall for a certain date, a student
> can register for an exam, etc. In this way, classes describe the behavior
> of object

Objects or Instances:

An object is distinguished by the fact it has it own properties and identity. Objects are always on a certain state which are expressed by the values of its attributes, meaning that it has a behavior described by a set of operations (identity property reflect at the operation set).

Encapsulation:

The encapsulation process is the protection over unauthorized access to the object´s internal state via a defined interface. For example `Java` has the following visibility markers:

```
public class Mystring
private class Myint
protected class Mychar
```

Those respectively permit access for all,
only within the object, and only for members of the same class, its subclasses,
and of the same package.

Messages:

The communication between objects is made up by *messages,* which represents a request to execute an operation. The operation is only done if the sender has the authorization to call up it.

This regulation can be done using *visibilities,* for example defining a set of parameters for the operations, for example the operation "+" can be have different meanings depending upon the parameter (the data type) integer and strings.

```
1+1=2 // Integers are added
"1" + "1" = "11"
"a" + "b" = "ab" //Strings are concatenated
```

Inheritance:

This is a mechanism for deriving new classes from existing ones. The *subclasses* derived from an existing one (superclass) inherits all the attributes and operations (specification and implementation) of the superclass.

Subclasses can:

- **Define new attributes and or operations**
- **Overwrite the implementation of inherited operations**
- **Add its own code to inherited operations**

As a consequence of inheritance classes can have a hierarchy and are extensible, this makes the basis for objected-oriented system development.

A *class hierarchy* consists of classes with similar properties, for example:

$Intern \leftarrow Employee \leftarrow Boss \leftarrow ...$

where $A \leftarrow B$ means that B is a subclass of A.

Polymorphism:

The prefix *poly* stands for "multiple" or "many" and *morphism* comes from *morphos* which means "forms" or "states". Thus the *polymorphism* is the ability to adopt different forms.

In a program execution, a polymorphic attribute can have references to objects from different classes.

A polymorphic operation can be executed on different classes and have different semantics in each case.

Implementation of this scenario can be done in certain ways like:

1. ***Parametric polymorphism* - use of type parameters**
2. ***Inclusion polymorphism* - application of operations to its classes and to their indirect and direct subclasses**
3. ***Overloading of operations***
4. ***Coercion***

Chapter 2 Notes:

A tour of UML (Unified modelling language):

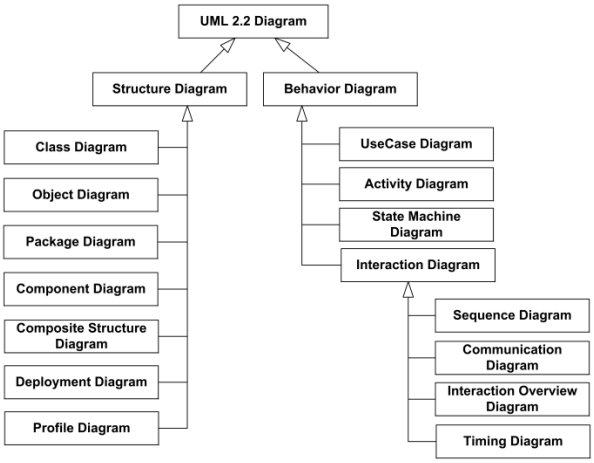
Usage:

UML is a object oriented modelling language which has usages through the entire software development process, favoring an iterative and incremental process through their diagrams.

Diagrams:

In UML a model is represented by diagrams graphically to provide a view of the reality or stage of the software development. The current version of UML has 14 diagrams that describe the structure or the behavior of the system.

Note that exists a distinction between structure and behavior diagrams in the figure aside:

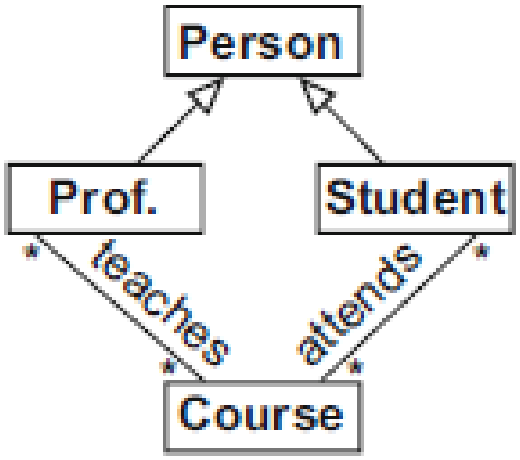


Structure Diagrams:

These types of diagrams represents the fundamentals and core properties/relationships of the system, meaning that those don't consider the dynamical behavior of it's elements. Some examples of structural diagrams are:

Class diagram:

Classes diagrams are extensively used through the modelling of systems for it's precision and correspondence with the Object Oriented Paradigm of software development and the conceptual data modelling of database design principles such as: generalization, inheritance, classes and association relationships.



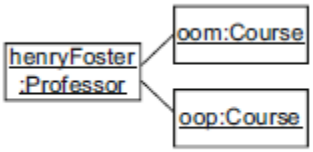
See that on the diagram aside we can see the class Person which is the common property of the classes Professor and Student.

This example shows us a generalization relationship between the class Person and it's members.

Object Diagram:

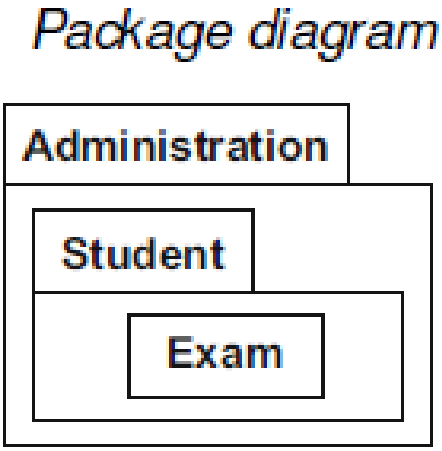
This diagram shows us the execution of a system state on a specific timeframe, meaning that it can give us concrete modelling of a system operations and object relations.

Note that the Professor represented as the object henryfoster teaches the courses Object-Oriented Modeling (oom) and Object-Oriented Programming (oop).



Package Diagram:

Packages diagrams are grouping models that are used to according to the common elements or properties of the given functional schema. We should note that these types of diagrams are most commonly integrated to other diagrams for further reference.

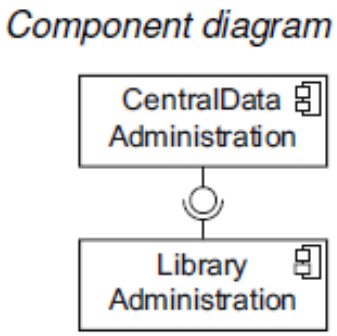


> **For example, in a university administration system, you could introduce packages that contain information about the teaching, the research, and the administrative aspects. Packages are often integrated in other diagrams rather than being shown in separate diagrams.**

The Component Diagram:

As UML aims to unify the modelling languages and approaches on a single oriented and organisational framework, it also attains to model the components of software development using these types of diagrams.

> A component is an independent, executable unit that provides other components with services or uses the services of other components.



The Composition Structure Diagram