UNIT-I

Concepts of Software Modeling

Contents

- Why model ?
- Principles of modeling
- What is UML ?
- Conceptual Model of the UML
  - Building Blocks
  - Rules
  - Common Mechanisms

Why Model ?

- Analyse the problem-domain
  - simplify reality
  - capture requirements
  - visualize the system in its entirety
  - specify the structure and/or behaviour of the system
- Design the solution
  - document the solution - in terms of its structure, behaviour, etc.

Principles of Modeling

- Choose your model well - *the choice of model profoundly impacts the analysis of the problem and the design of the solution.*
- Every model may be expressed at different levels of precision - *the same model can be scaled up (or down) to different granularities.*
- The best models are connected to reality - *simplify the model, but don't hide important details.*
- No single model suffices - *every nontrivial system has different dimensions to the problem and its solution.*

What is UML ?

- UML - Unified Modeling language
- UML is a modeling language, not a methodology or process
- Fuses the concepts of the Booch, OMT, OOSE methods
- Developed by Grady Booch, James Rumbaugh and Ivar Jacobson at Rational Software.
- Accepted as a standard by the Object Management Group (OMG), in 1997.

More on UML...

UML is a modeling language for visualising, specifying, constructing and documenting the artifacts of software systems.

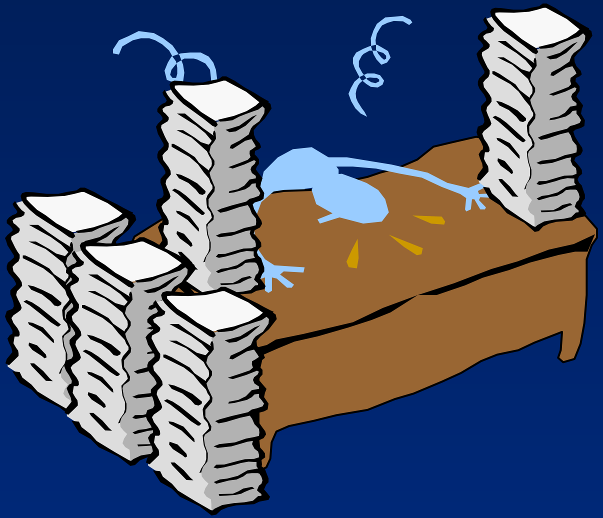
Visualising - *a picture is worth a thousand words; a graphical notation articulates and unambiguously communicates the overall view of the system (problem-domain).*

More on UML...

Specifying - *UML provides the means to model precisely, unambiguously and completely, the system in question.*

Constructing - *models built with UML have a "design" dimension to it; these are language independent and can be implemented in any programming language.*

More on UML...



Documenting - *every software project involves a lot of documentation - from the inception phase to the deliverables.*

Documentation is (among others) for:

- Requirements
- Design
- Tests

UML provides the notations for documenting some of these artifacts

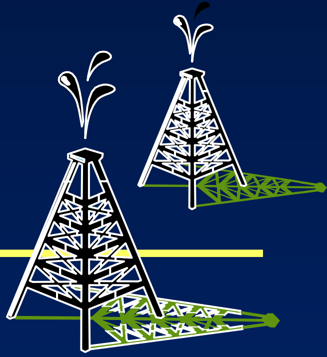
Conceptual Model of UML

- Building Blocks
  - *Things*
  - *Relationships*
  - *Diagrams*
- Rules
- Common Mechanisms
  - *Specifications*
  - *Adornments*
  - *Common Divisions*
  - *Extensibility Mechanisms*

UML Building Blocks

- Things
  - *Structural*
  - *Behavioral*
  - *Grouping*
  - *Annotational*
- Relationships
  - *Dependency*
  - *Association*
  - *Generalisation*
  - *Realization*
- Diagrams
  - *Class Diagram*
  - *Object Diagram*
  - *Use Case Diagram*
  - *Behaviour Diagram*
  - *Implementation Diagram*

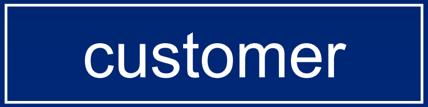
Structural Things



The nouns of UML models; usually the static parts of the system in question.

- Class - *an abstraction of a set of things in the problem-domain that have similar properties and/or functionality.*

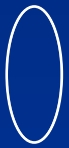
  Notation:

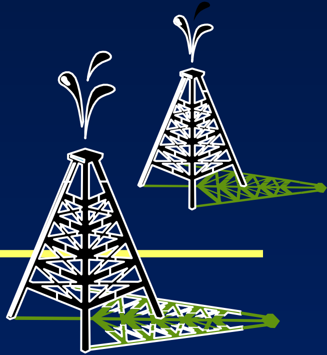
  customer

- Interface - *a collection of operations that specify the services rendered by a class or component.*

  Notation:

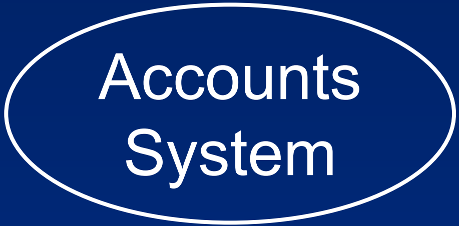
  

Structural Things (contd.)



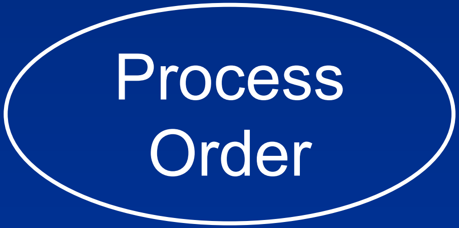
- Collaboration - *a collection of UML building blocks (classes, interfaces, relationships) that work together to provide some functionality within the system.*

Notation:

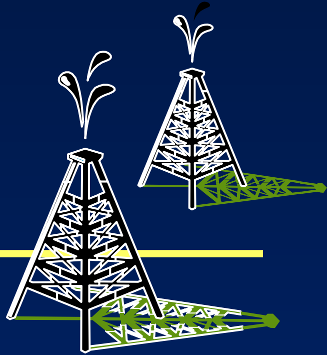


- Use Case - *an abstraction of a set of functions that the system performs; a use case is "realized" by a collaboration.*

Notation:

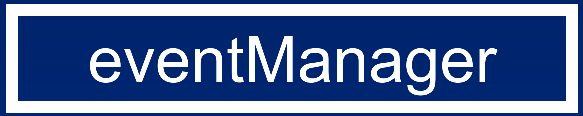


Structural Things (contd.)



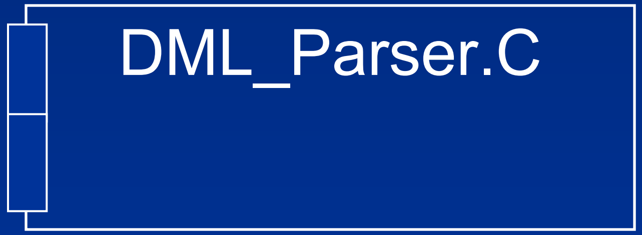
- Active Class - *a class whose instance is an active object; an active object is an object that owns a process or thread (units of execution)*

Notation:

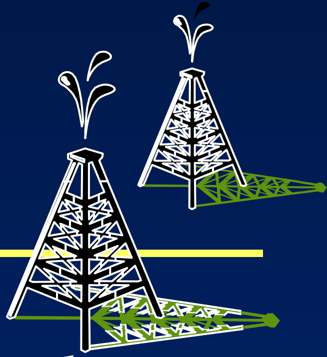


- Component - *a physical part (typically manifests itself as a piece of software) of the system.*

Notation:

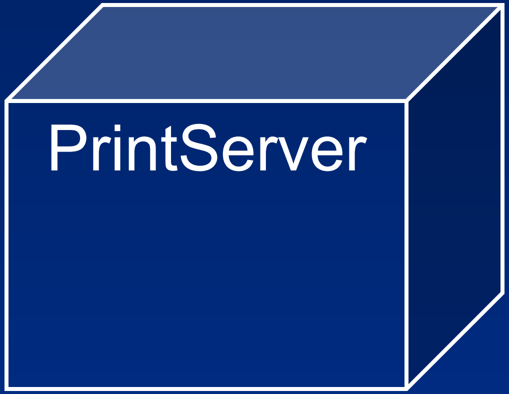


Structural Things (contd.)

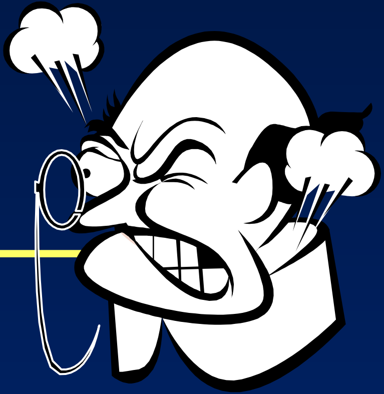


- Node - *a physical element that exists at run-time and represents a computational resource (typically, hardware resources).*

Notation:



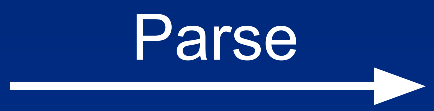
Behavioral Things



The verbs of UML models; usually the dynamic parts of the system in question.

- Interaction - *some behaviour constituted by messages exchanged among objects; the exchange of messages is with a view to achieving some purpose.*

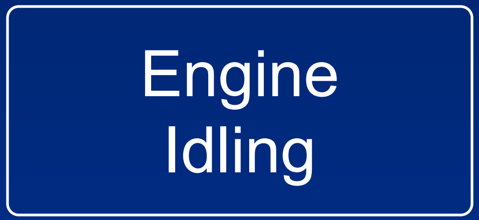
Notation:



Behavioral Things (contd.)

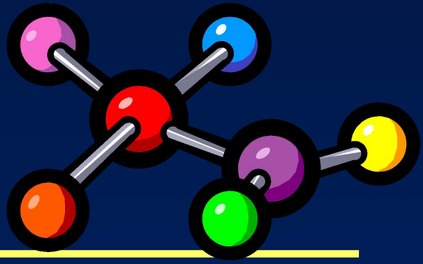
- State machine - *a behaviour that specifies the sequence of "states" an object goes through, during its lifetime. A "state" is a condition or situation during the lifetime of an object during which it exhibits certain characteristics and/or performs some function.*

Notation:



Engine Idling

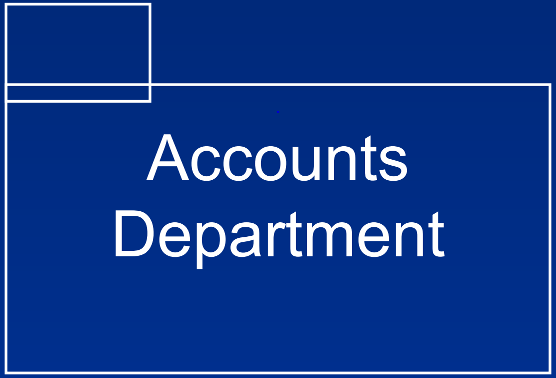
Grouping Things



The organisational part of the UML model; provides a higher level of abstraction (granularity).

- Package - *a general-purpose element that comprises UML elements - structural, behavioral or even grouping things. Packages are conceptual groupings of the system and need not necessarily be implemented as cohesive software modules.*

Notation:

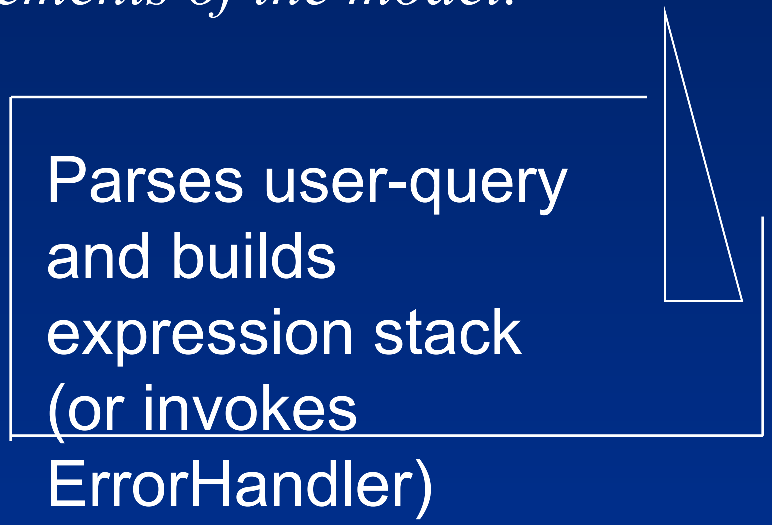


Annotational Things

The explanatory part of the UML model; adds information/meaning to the model elements.

- Note - *a graphical notation for attaching constraints and/or comments to elements of the model.*

Notation:



Parses user-query and builds expression stack (or invokes ErrorHandler)

Relationships

Articulates the meaning of the links between things.

- Dependency - *a semantic relationship where a change in one thing (the independent thing) causes a change in the semantics of the other thing (the dependent thing).*

  Notation: ⟶

  (arrow-head points to the independent thing)

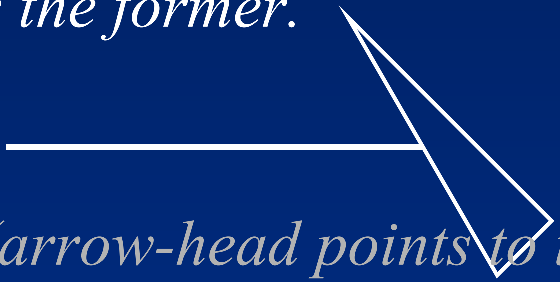
- Association - *a structural relationship that describes the connection between two things.*

  Notation: ———

Relationships (contd.)

- Generalisation - *a relationship between a general thing (called "parent" or "superclass") and a more specific kind of that thing (called the "child" or "subclass"), such that the latter can substitute the former.*

Notation:



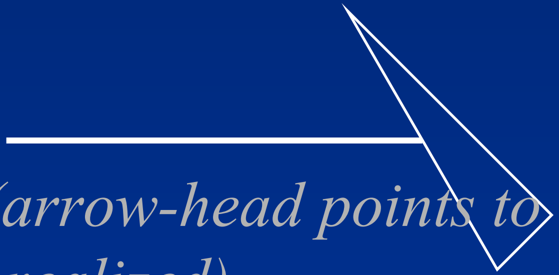
(arrow-head points to the superclass)

Relationships (contd.)

- Realization - *a semantic relationship between two things wherein one specifies the behaviour to be carried out, and the other carries out the behaviour.*
  - " a collaboration *realizes* a Use Case"

    the Use Case specifies the behaviour (functionality) to be carried out (provided), and the collaboration actually implements that behaviour.

Notation:



(arrow-head points to the thing being realized)

Diagrams

The graphical presentation of the model. Represented as a connected graph - vertices (things) connected by arcs (relationships).

UML includes nine diagrams - each capturing a different dimension of a software-system architecture.

- Class Diagram
- Object Diagram
- Use Case Diagram
- Sequence Diagram
- Collaboration Diagram
- Statechart Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram

More on Diagrams...

- Class Diagram - *the most common diagram found in OOAD, shows a set of classes, interfaces, collaborations and their relationships. Models the static view of the system.*
- Object Diagram - *a snapshot of a class diagram; models the instances of things contained in a class diagram.*
- Use Case Diagram - *shows a set of "Use Cases" (sets of functionality performed by the system), the "actors" (typically, people/systems that interact with this system[problem-domain]) and their relationships. Models WHAT the system is expected to do.*

More on Diagrams...

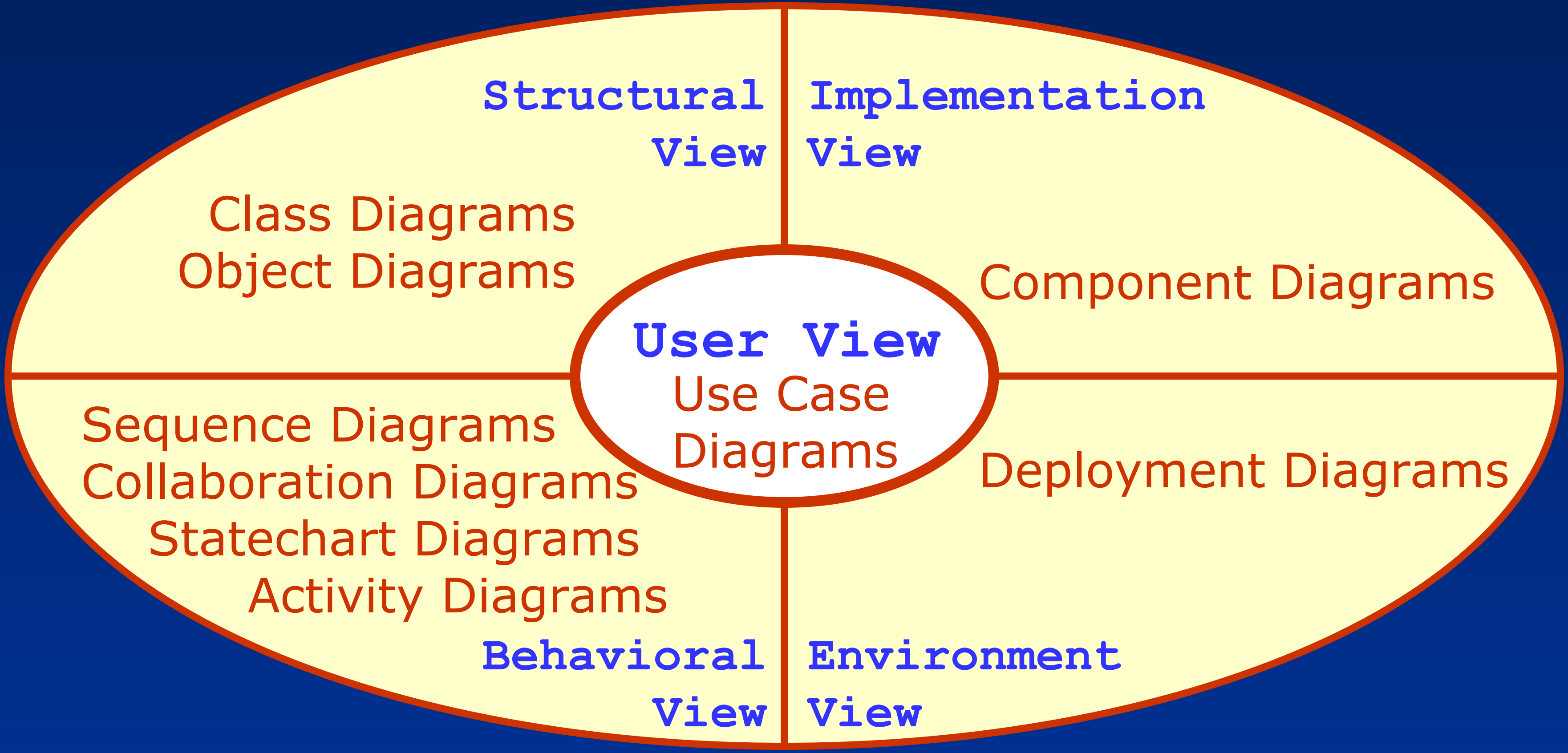
- Sequence Diagram - *models the flow of control by time-ordering; depicts the interaction between various objects by of messages passed, with a temporal dimension to it.*
- Collaboration Diagram - *models the interaction between objects, without the temporal dimension; merely depicts the messages passed between objects.*
- Statechart Diagram - *shows the different state machines and the events that leads to each of these state machines. Statechart diagrams show the flow of control from state to state.*

More on Diagrams...

- Activity Diagram - *shows the flow from activity to activity; an "activity" is an ongoing non-atomic execution within a state machine.*
- Component Diagram - *shows the physical packaging of software in terms of components and the dependencies between them.*
- Deployment Diagram - *shows the configuration of the processing nodes at run-time and the components that live on them.*

Dimensions...

. . .of Software Architecture



Structural View

Class Diagrams
Object Diagrams

Implementation View

Component Diagrams

User View
Use Case Diagrams

Sequence Diagrams
Collaboration Diagrams
Statechart Diagrams
Activity Diagrams

Behavioral View

Deployment Diagrams

Environment View

Rules

- Specify what a well-formed model should look like.
- The UML has semantic rules for
  - Names
  - Scope
  - Visibility
  - Integrity
  - Execution

Common Mechanisms

- Mechanisms/elements that apply consistently throughout the language:
  - Specifications
  - Adornments *(Notes)*
  - Common Divisions
  - Extensibility Mechanisms
    - *Stereotypes*
    - *Tagged values*
    - *Constraints*

Adornments

“Adorn” the model - i.e., enhance the model. Adds to the meaning and/or semantics of the element to which it pertains.

“Notes” are the mechanism provided by UML for adorning a model:

- graphical symbol to render constraints, comments, etc.
- a note that renders only a comment has no semantic impact on the element it is adorning; at most adds meaning to it and/or provides guidelines for implementation.

Stereotypes

- Used to create new building blocks from existing blocks.
- New building blocks are domain-specific.
- A particular abstraction is marked as a “stereotype” and this stereotype is then used at other places in the model to denote the associated abstraction.

Notation: «metaclass»

Tagged Values

- Used to add to the information of the element (not of its instances).
- Stereotypes help create new building blocks; tagged values help create new attributes.
- Commonly used to specify information relevant to code generation, configuration management, etc.

Notation: {version=1.4}

Constraints

- Used to create rules for the model.
- Rules that impact the semantics of the model, and specify conditions that must be met.
- Can apply to any element in the model - attributes of a class, relationship, etc.

Notation: { incomplete, disjoint }