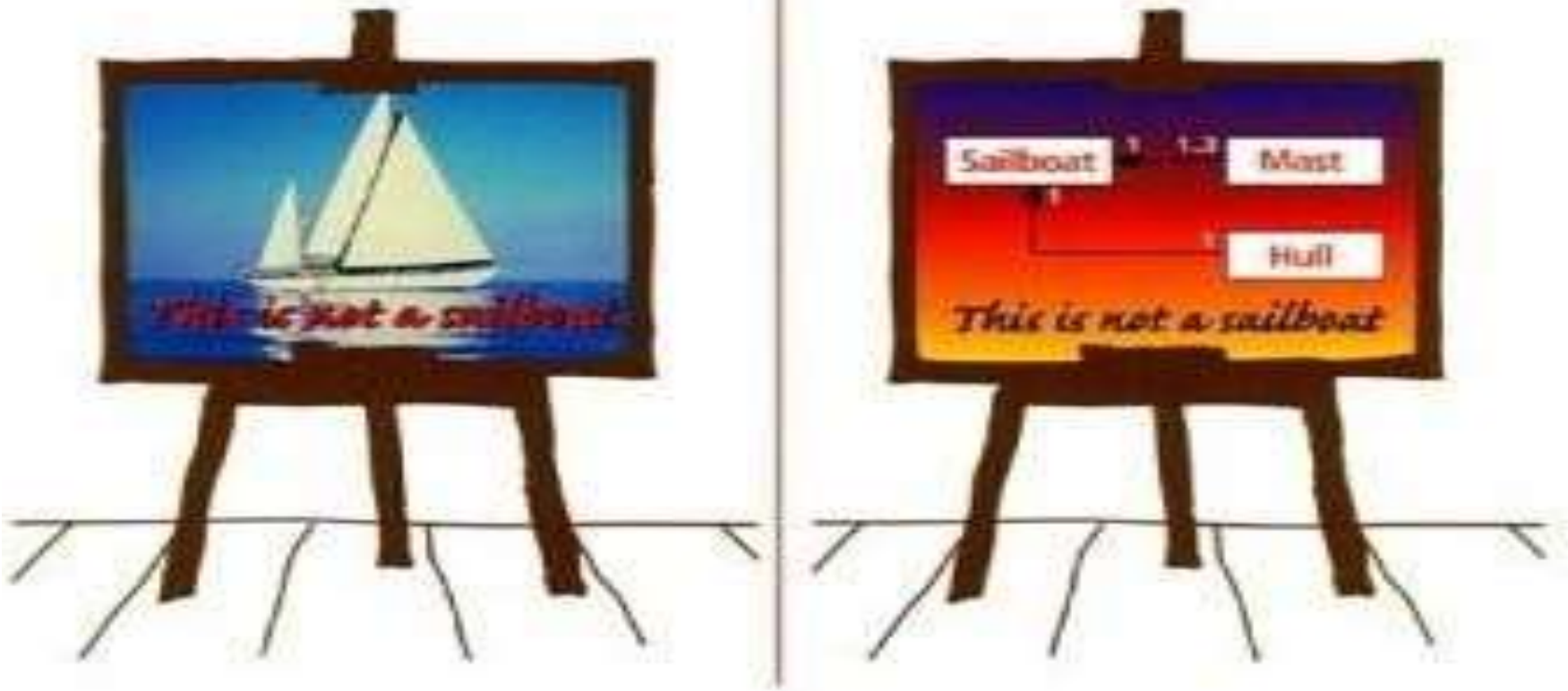
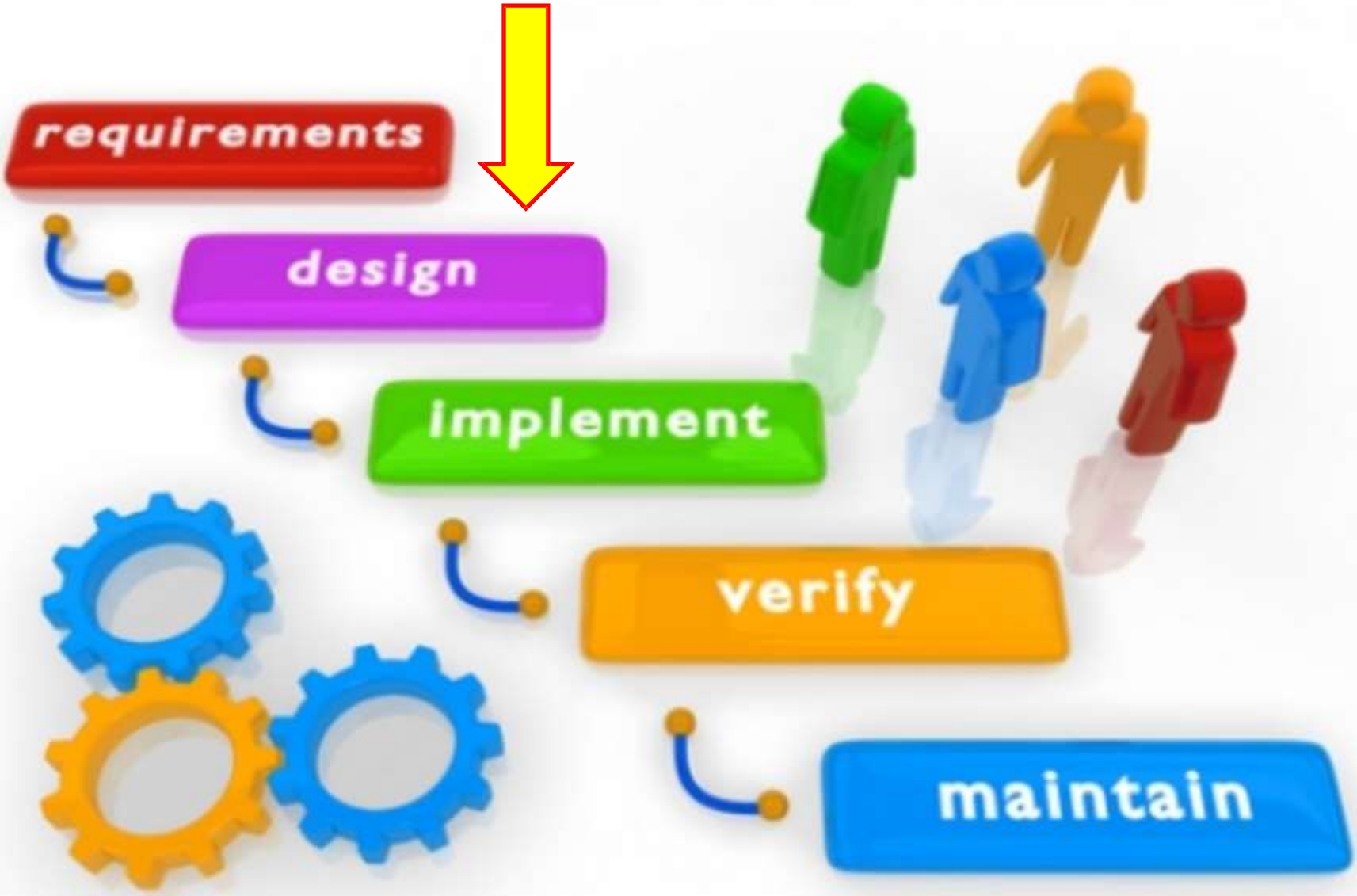


Unit 4. Object-Oriented Modeling and Design

4.1 Introduction

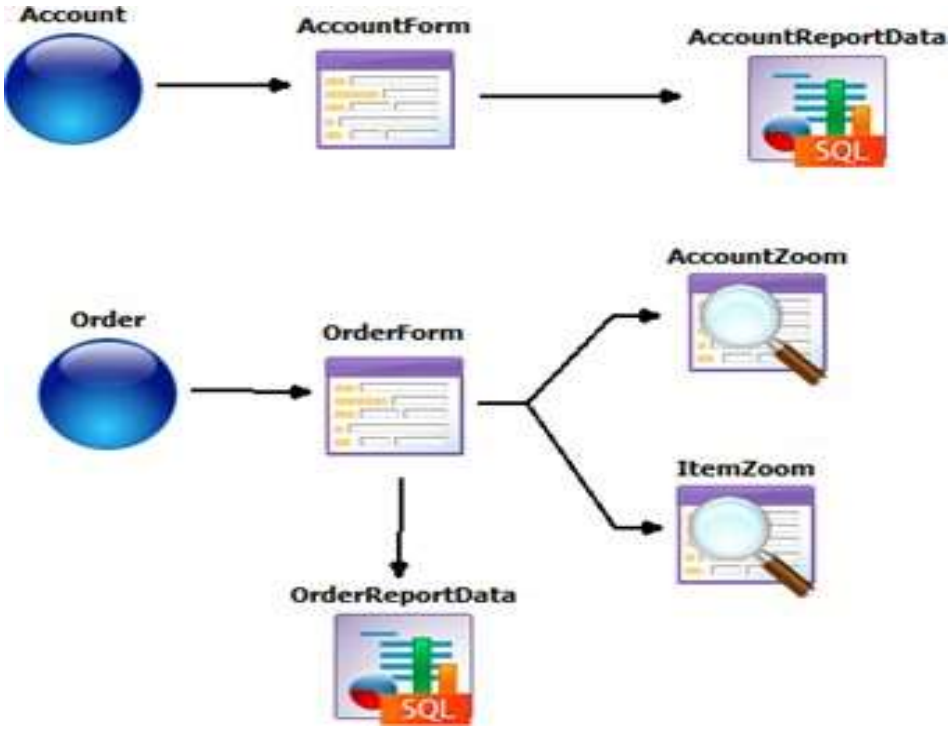
Contents

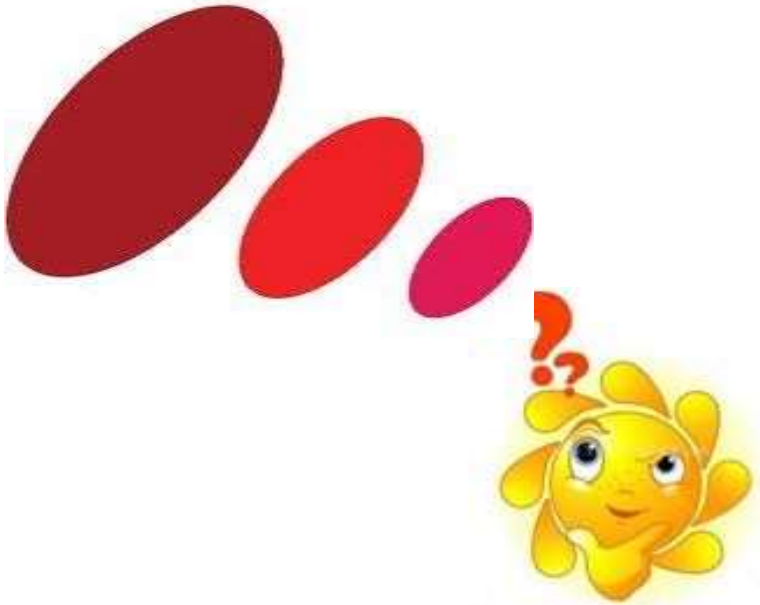


SOFTWARE ENGINEERING

Introduction

- **OO modeling and design** is a way of thinking about problems using models organized around real world concepts





Introduction …

- Fundamental construct is the **object**, which combines both data structure and behavior
- **OO modeling and design promotes**
  - Better understanding of requirements

- Analysis→Design→Implementation

object-oriented notations & process

OO characteristics

1) Identity

- Data is quantized into **discrete, distinguishable** entities called **objects**

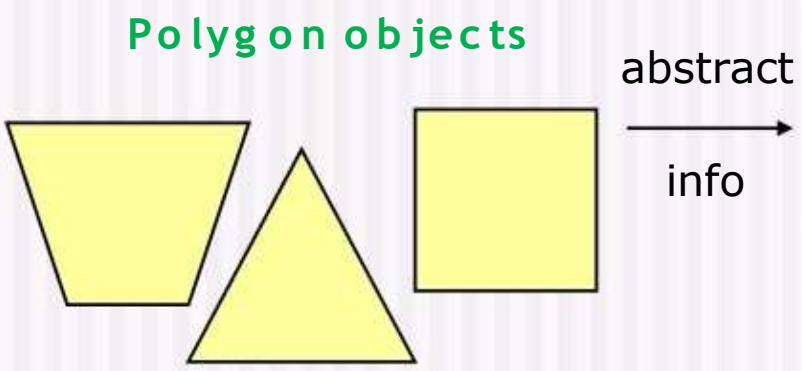
- Each object has is own inherent identity
  - Two objects are **distinct** even if all their attribute values are **identical**

- Each object has a **unique** handle by which it can be referenced
  - Handle in various ways such as an address, array index, or artificial number

OO characteristics (2/4)

Classification

- Objects with the same data structure (attributes) and behaviour (operations) are grouped into a **class**
- A Class is an abstraction that describes properties important to an application and ignores the rest
- Each class describes an infinite set of individual objects
- An Object is an instance of a class



Objects and Classes

Bicycle objects

abstract
into

Bicycle class

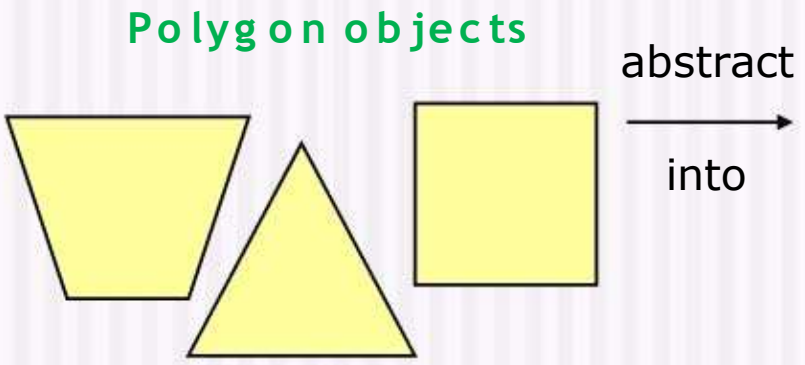
Attributes
- frame size
- wheel size
- number of gears
- material

Operations
- shift
- move
- repair

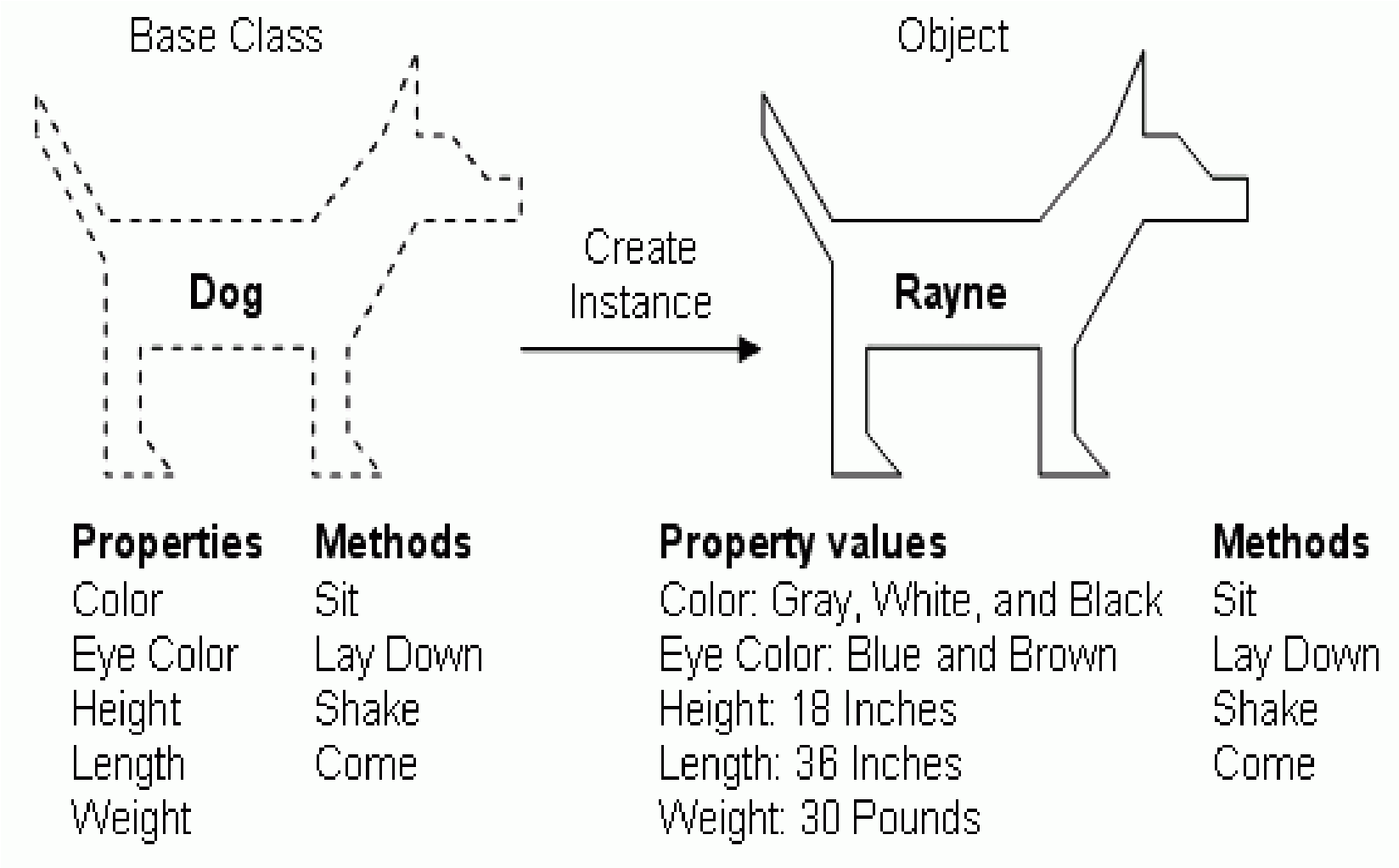
OO characteristics

2) Classification

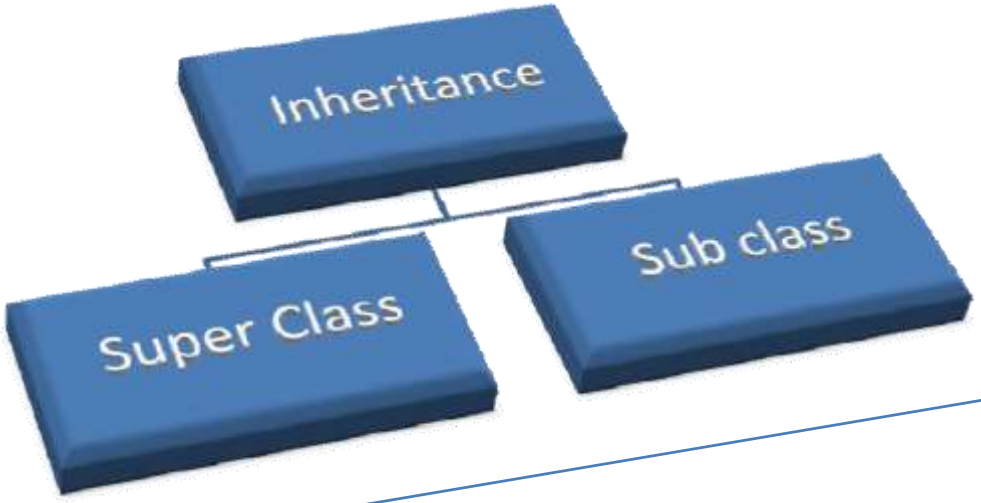
- Objects with the same data structure (attributes) and behaviour (operations) are grouped into a class
- A Class is an abstraction that describes properties important to an application and ignores the rest
- Each class describes an infinite set of individual objects
- **An Object is an instance of a class**



An Object is an instance of a class



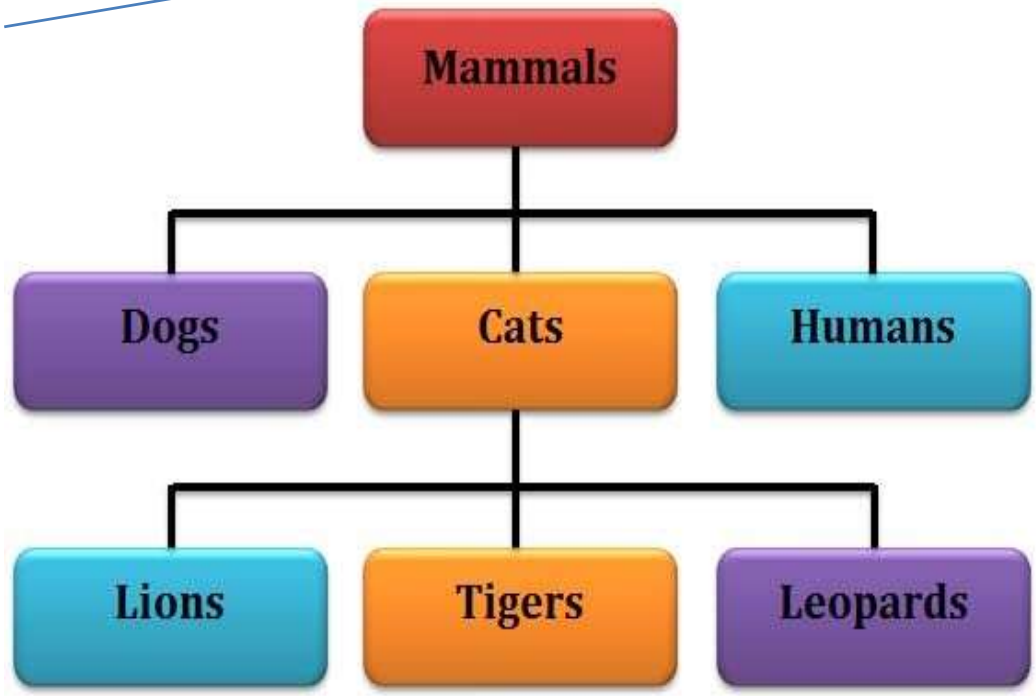
3) Inheritance



Sharing of attributes and operations (features) among classes based on a **hierarchical relationship**

- **Super class** has **general information** that **subclasses refine** and **elaborate**
- Each **subclass** incorporates, or inherits, all the features of its super class and adds its own **unique features**

  Greatly reduce repetition
- Ability to factor out **common features** of several classes into a super class



The simple hierarchy of Mammals

4) Polymorphism

- **Same operation** behaves differently for different classes
- An operation is a **procedure** or **transformation** that an object performs
- **Method**
  - An implementation of an operation by a **specific class**
  - Each object " **knows how**" to perform its own operation.
  - OO operator is polymorphic

OO Development

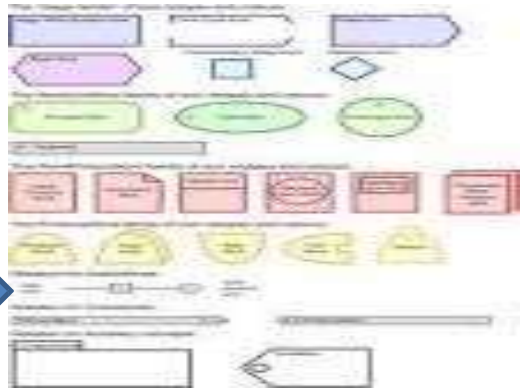
- **Essence of OO development**
  - Identification and organization of application concepts, rather than their final representation in **programming language.**

1) Modeling Concepts, Not Implementation

- Focus on analysis and design
- Encourages software developers to **work and think**
- Should be **identified**, **organized**, and **understood**
  - Premature focus on implementation restricts **design choices**
  - **Design flaws** during implementation costs more and leads to inferior product
- **Conceptual process** independent of a programming language.

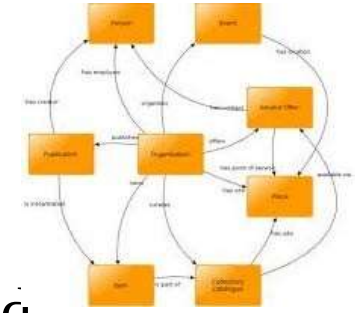
  OO is fundamentally a way of thinking and not programming techniques.

OO development



2) OO Methodology

- Process for OO development with graphical notation (OO Concepts)
- Methodology = building a model + adding details during design
- Same notation is used from
  - analysis ➡ design ➡ implementation.
  - **Information** added in one stage is not lost and transformed to next stage



Methodology Stages

- System Conception
- Analysis
- System Design
- Class Design
- Implementation

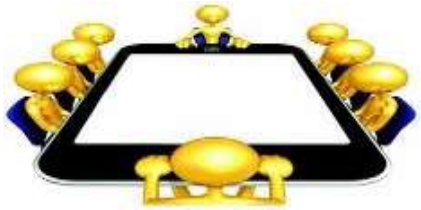
Methodology Stages

i) System conception

s/w development begins with **business analyst** and formulate **tentative requirements**

ii) Analysis

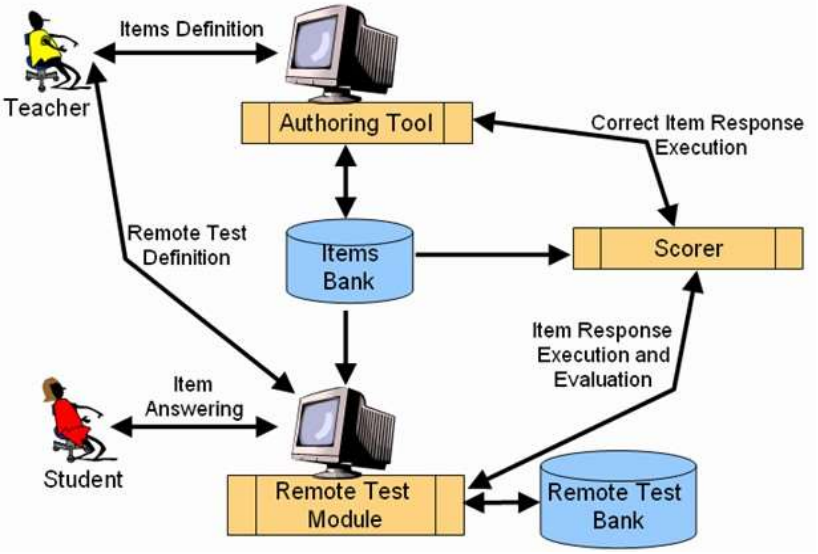
- Restates the requirements from system conception by constructing models
- Analyst must work with the **requestor** to understand the **problem statements**
- Analysis model (abstract) describes **what the system must do**, and **not how it will do.(no implementation decisions)**



Domain model- description of all the module related to given problem

Application model- description about a specific task(visible to the user)

- **Application experts** who are not a programmer can understand & criticize good model



iii) System Design

- **System architecture** – solving the application problems

- **System designer decides**

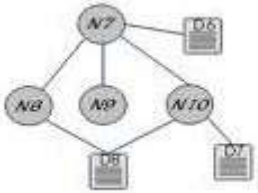
  – what performance characteristics to optimize

  – Choose a strategy of attacking the **problems.**

  – Make tentative resource allocation.

iv) Class Design

- Add details to the **analysis model** (based on system design strategy)
- Class designer elaborates both **domain** and **application objects** using the same OO concepts & notation.
- Focus is to implement the **data structure** and **algorithm**.



v) Implementation

- Implementer translate class and relationship

Programming language, DB, H/W

- Programming should be straight forward (**hard decision** are already made)

- Follow good software engineering practice (System remains flexible & extensible)

OO Development

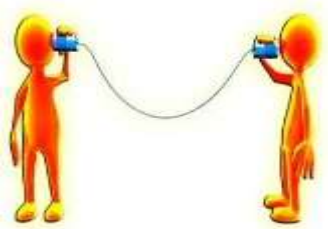
3) Modeling

A model is a simplification of reality

- **Abstraction** for the purpose of understanding before building it

- **Isolate** those aspects which are important and **suppress** the rest(unimportant)

Purpose



- Testing a physical entity before building it
- Communication with customers
- Visualization
- Reduction of complexity

Design a System

- **Class** — For the objects in the system and their relationship
- **State** — For the life history of the object
- **Interaction** — For the interaction among the objects

OO development

4) Three models

Class model

- Function
  - Describes the **static structure** of the object in the system – identity, relationship to other object, attributes, operations
  - "**data**" aspects of the system
- Goal
  - Provides context for **state** and **interaction model**
  - Capture **important concepts** of an application from the real world
- Representation
  - Class diagrams → Graph
    - **Nodes**: Class
    - **Arc:** relationship B/W Classes
  - Generalization, aggregation

Three models (Cont'd)

State model

- Function
  - Describes **objects' time** and **sequencing** of operation
- Goal
  - Capture "**control"** aspect of system that describes the **sequences of operations** that occur
- Representation
  - State diagrams

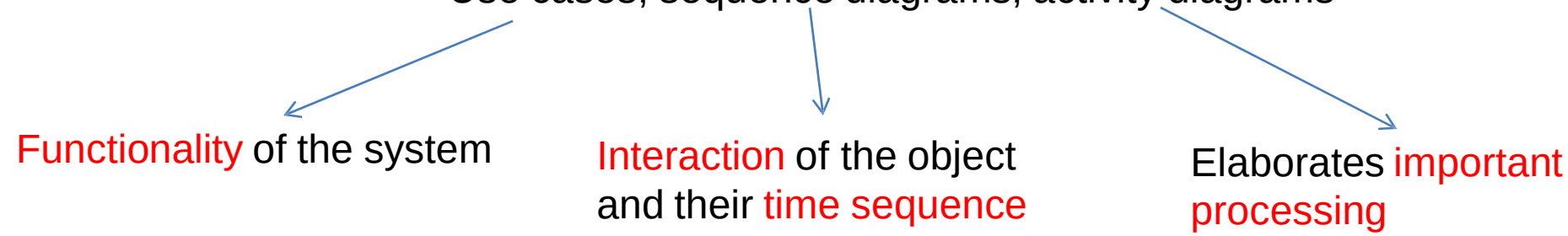
Graph :

nodes-states ; **arcs**- transition between states

Three models (Cont'd)

Interaction model

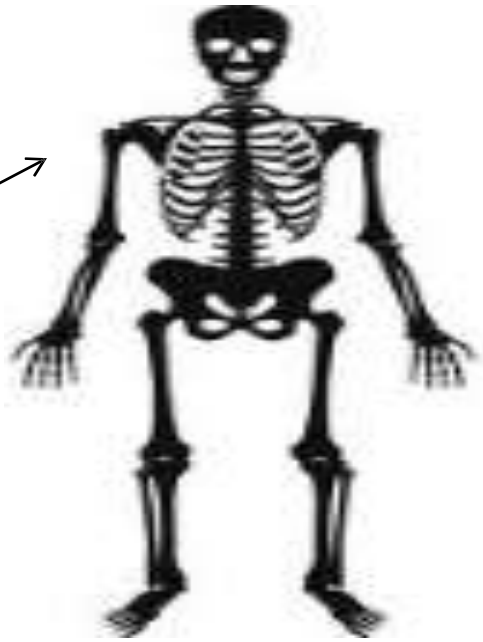
- Function
  - Describes **interactions** between objects
  - Individual objects **collaborate** to achieve the behavior of the whole system
- Goal
  - **Exchanges** between objects and provides a holistic overview of the operation of a system
- Representation
  - Use cases, sequence diagrams, activity diagrams



OO Themes

1) Abstraction

Just like a skeleton. You can fit anything on it you like.



•Focus on **essential aspects** of an application while ignoring the details

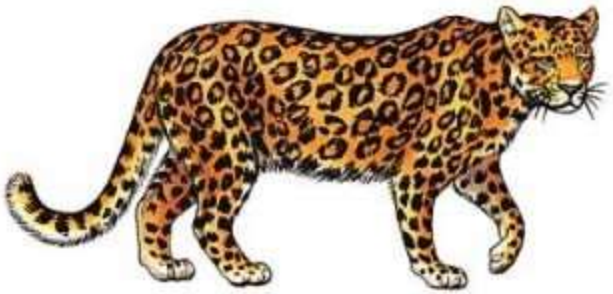
•What an object is and does, before deciding how to implement

•Preserves the freedom to make decision as long as possible by avoiding premature commitments to details

Example

- A class called Animal.
- It has properties like ears,colour, eyes but they are not defined.
- It has methods like Running(), Eating(), etc. but the method does not have any body
- all animals will have the above properties and methods but you decide how to do them.
- sub class of the class Animal called **Tiger.**

Color is **yellow**



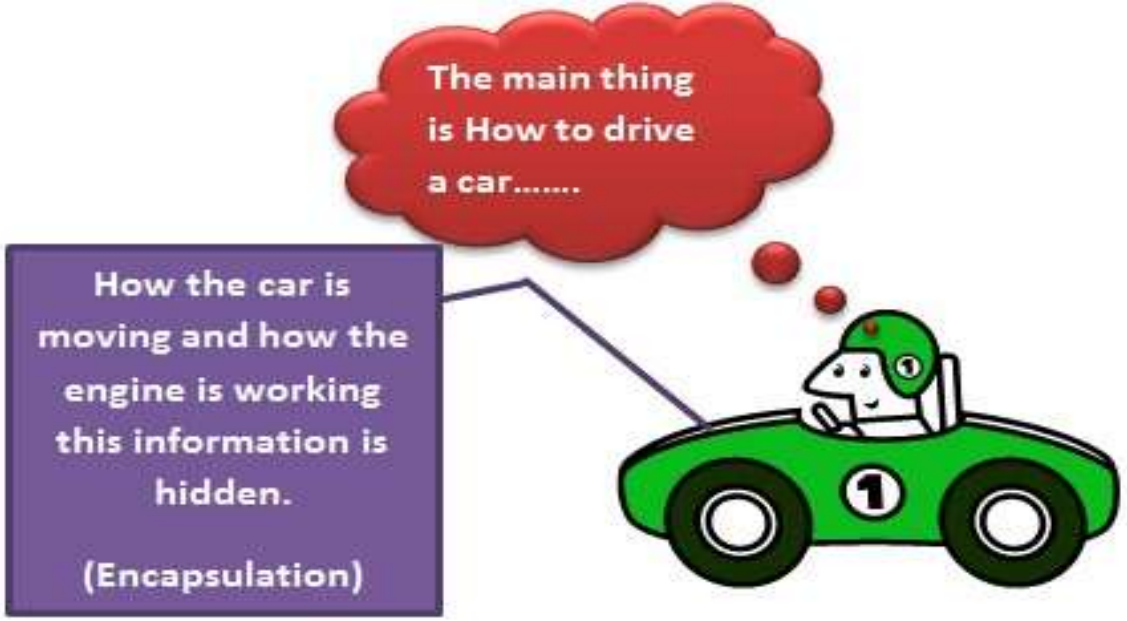
running is very **fast**

color is **black**

running is very **slow**

OO themes

2) Encapsulation



- Separates the external aspects of an object from internal implementation
- Data structure and behaviour is encapsulated in a single entity
- Ensuring reliability and maintainability

  - Information exchange is done by public interface among objects
  - Change internal data structure does not affect other objects

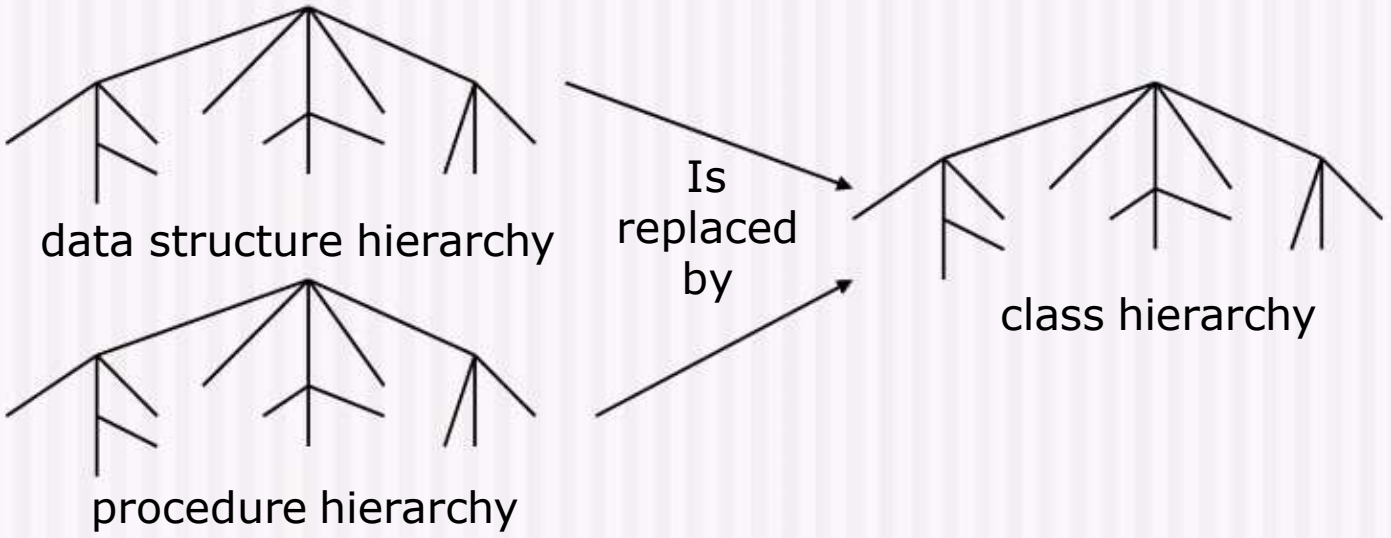
Example

- capsule that the doctor gives us
- We just have to take the capsule to get better
- **don't have to worry** about
  - what medicine is inside the capsule or
  - how it will work on our body.
- user does not have to worry how this **methods** and **properties** work.

OO themes

3) Combining data and behavior

Data structure hierarchy matches the operation inheritance hierarchy



OO themes

4) Sharing

No redundancy (Inheritance)

Reusability (tools- abstraction, inheritance, encapsulation)

5) Emphasis on the essence of an object

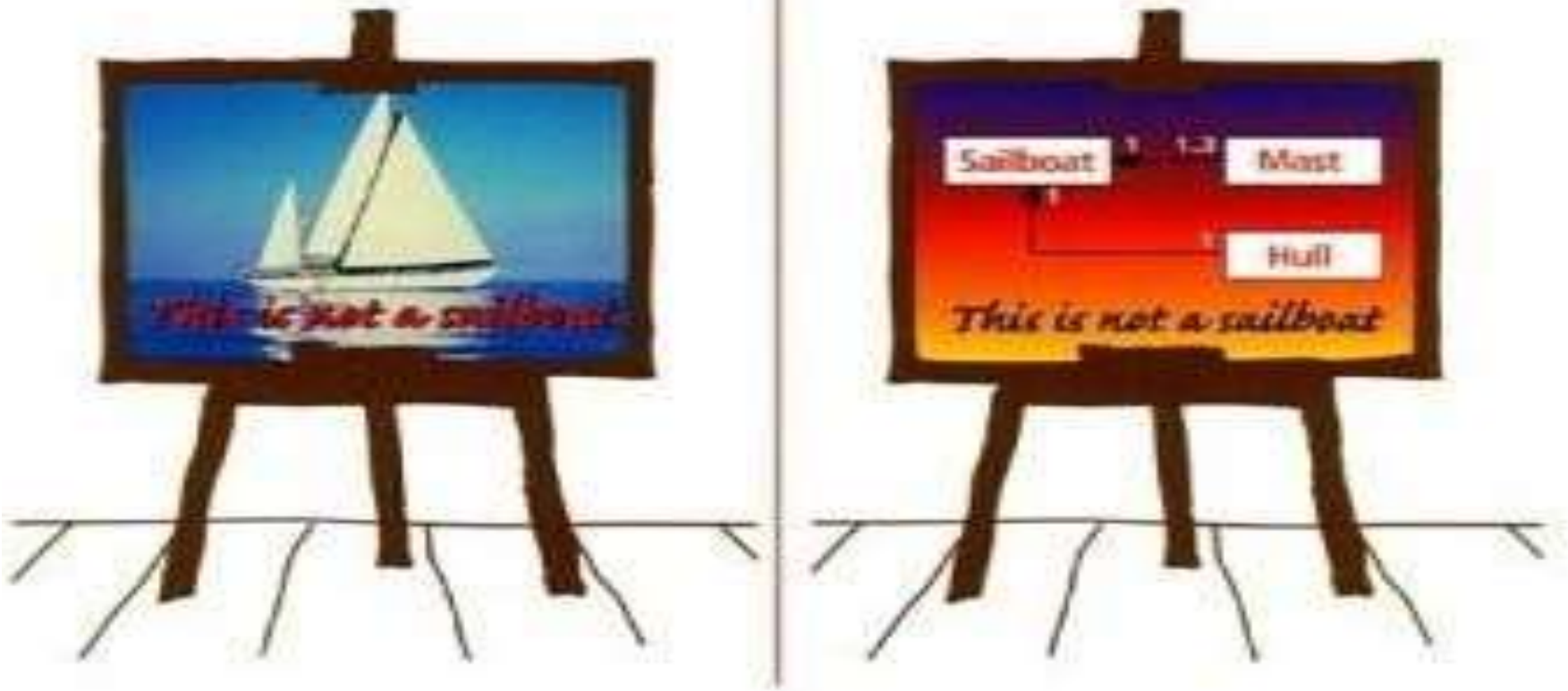
Focus on **what an object is**

- Rather than how it is used

6) Synergy

Identity, classification, polymorphism, inheritance

- Be cleaner, more general and robust



Unit 4. Object-Oriented Modeling and Design

4.2 Modeling as a Design Technique

Contents

Introduction- Modeling

Abstraction

Three

Models

Introduction

- **A model**
  - An abstraction of something for the purpose of understanding it before building it
  - Easier to manipulate than the original entity, because a model omits nonessential details
  - Engineers, artists, and craftsman have built models for thousand of years to try out designs before executing them
- **To build complex system**
  - Developer must abstract different views of the systems, build models using precise notations, verify that the models satisfy the requirements of the system, and gradually add detail to transform the model into an implementation.

Modeling

- **Purposes**
  - Testing a physical entity before building it
  - Communication with customers
  - Visualization
  - Reduction of complexity
- **A good model**
  - Captures the crucial aspects of a problem and omits the others
  - A model that contains extraneous detail unnecessarily limits your choice of design decisions and diverts attention from the real issues.

Object Modeling Technique (OMT)

- **Three views of modeling systems**
  - Object model
    - static, structural, "data“ aspects of a system
  - Dynamic model
    - temporal, behavioral, "control“ aspects of a system
  - Functional model
    - transformational, "function“ aspects of a system
- **Typical software procedure**
  - It uses data structures (object model)
  - It sequences operations in time (dynamic model)
  - It transforms values (functional model)

Three Models of OMT (rev.)

- **Class model**
  - Describe static structure of objects in system and relationships
  - Contain class diagrams which is a graph
    - nodes: object classes, arcs: relationships among classes
- **State model**
  - Describe aspects of a system that change over time
  - Specify control aspect of system
  - Contain state diagrams which is a graph
    - nodes: states, arcs: transition between states caused by events
- **Interaction model**
  - Describe data value transformation within system
  - Contain use cases, sequence diagrams and activity diagrams

UNIFIED MODELING LANGUAGE -UML

Introduction to UML

•UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

•Rumbaugh joined Booch at Rational in 1994; in 1995, Rational added Jacobsen to their team. In 1996, work on the UML was begun.

•In January of 1997, Rational released UML 1.0 to the OMG as their proposal for a methods standard

•It was initially started to capture the behavior of complex software and non-software system and now it has become an OMG standard.

Introduction to UML

- UML stands for Unified Modeling Language.
- different from the other common programming languages such as C++, Java, COBOL, etc.
- pictorial language used to make software blueprints.
- a general purpose visual modeling language to visualize, specify, construct, and document software system.
- used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc.
- UML is not a programming language but tools can be used to generate code in various languages using UML diagrams.

A Conceptual Model of UML

•A conceptual model can be defined as a model which is made of concepts and their relationships.

•A conceptual model is the first step before drawing a UML diagram.

•It helps to understand the entities in the real world and how they interact with each other.

•The conceptual model of UML can be mastered by learning the following three major elements −

- •UML building blocks
- •Rules to connect the building blocks
- •Common mechanisms of UML

Object-Oriented Concepts

- **UML diagrams are representation of object-oriented concepts only.**
- **Following are some fundamental concepts of the object-oriented world –**
  - **Objects** – Objects represent an entity and the basic building block.
  - **Class** – Class is the blue print of an object.
  - **Abstraction** – Abstraction represents the behavior of an real world entity.
  - **Encapsulation** – Encapsulation is the mechanism of binding the data together and hiding them from the outside world.
  - **Inheritance** – Inheritance is the mechanism of making new classes from existing ones.
  - **Polymorphism** – It defines the mechanism to exists in different forms.

OO Analysis and Design

- **The purpose of OO analysis and design can described as –**
  - Identifying the objects of a system.
  - Identifying their relationships.
  - Making a design, which can be converted to executables using OO languages.
- **There are three basic steps where the OO concepts are applied and implemented. The steps can be defined as**
- **OO Analysis → OO Design → OO implementation using OO languages**

Building blocks of UML

- **The building blocks of UML can be defined as –**
  - Things
  - Relationships
  - Diagrams

Things

- **Things are the most important building blocks of UML. Things can be**
  - Structural
  - Behavioral
  - Grouping
  - Annotational

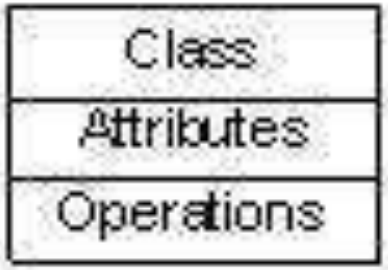
Structural Things

•Structural things define the static part of the model. They represent the physical and conceptual elements.

• Following are the brief descriptions of

•**Class** − Class represents a set of objects having similar responsibilities.

Class

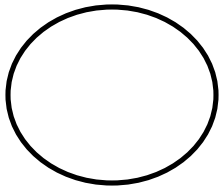


Window
origin size
open() close() move() display()

Structural Things

- **Interface** − Interface defines a set of operations, which specify the responsibility of a class.

Interface



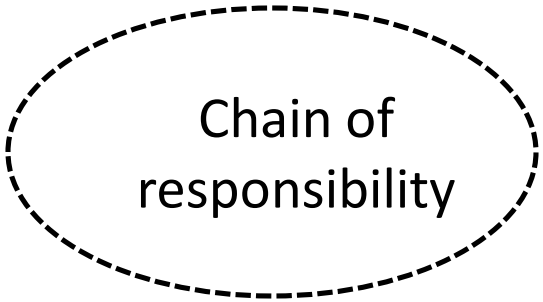
ISpelling

Structural Things

- **Collaboration** –Collaboration defines

an interaction between elements

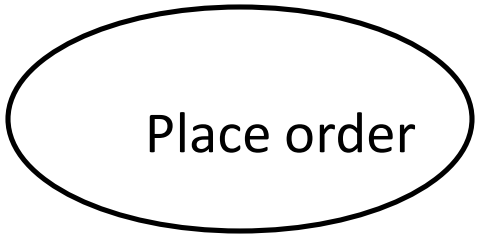
Collaboration



Structural Things

- **Use case** −Use case represents a set of actions performed by a system for a specific goal.

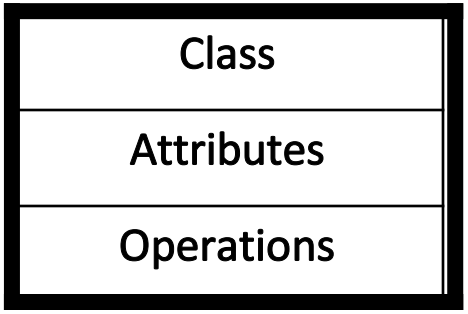
Use case

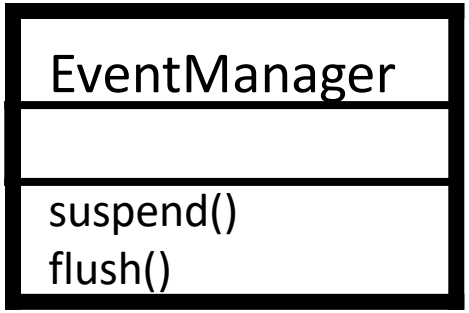


Structural Things

- **Active class**–Class whose objects own one or more processes or threads and therefore can initiate control activity.

Active class

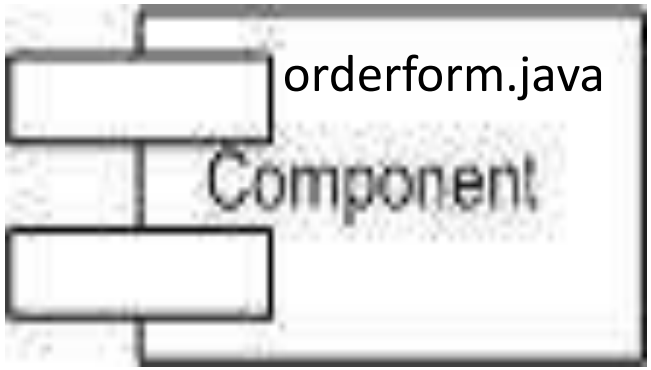




Structural Things

- **Component** –Component describes the physical part of a system

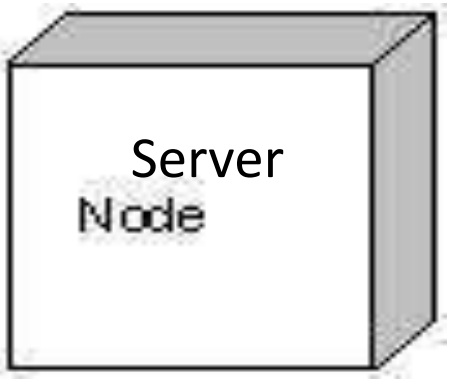
Component



Structural Things

- **Node** – A node can be defined as a physical element that exists at run time.

Node

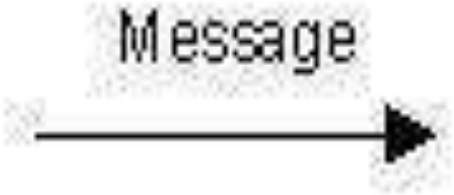


Behavioral Things

•A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things –

•**Interaction** – Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

Messages



display

Behavioral Things

- **State machine** – State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

States

Grouping Things

•Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

•**Package** − Package is the only one grouping thing available for gathering structural and behavioral things.

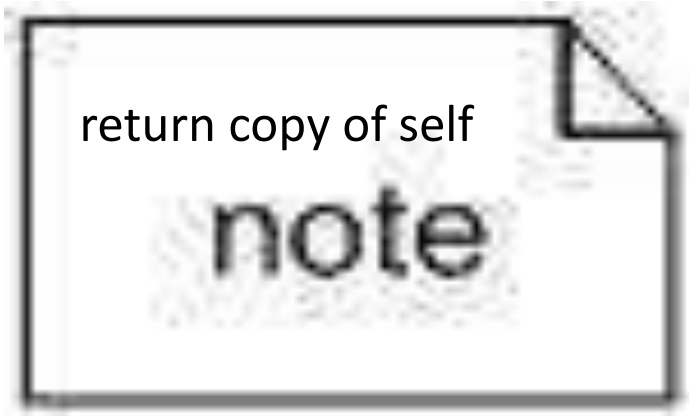
Package

Annotational Things

•Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

• **Note** - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element

Note



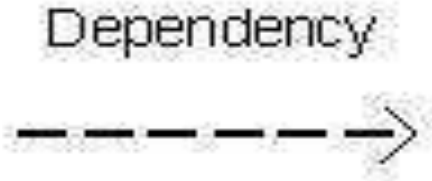
Relationships

- **Relationship is another most important building block of UML.**
- **It shows how the elements are associated with each other and this association describes the functionality of an application.**
- **There are four kinds of relationships available.**
  - Dependency
  - Association
  - Generalization
  - Realization

Relationships

- Dependency- Dependency is a relationship between two things in which change in one element also affects the other.

Dependencies



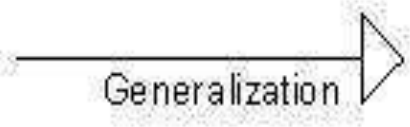
Relationships

- **Association**- Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.

Relationships

- **Generalization**- Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.

Generalization

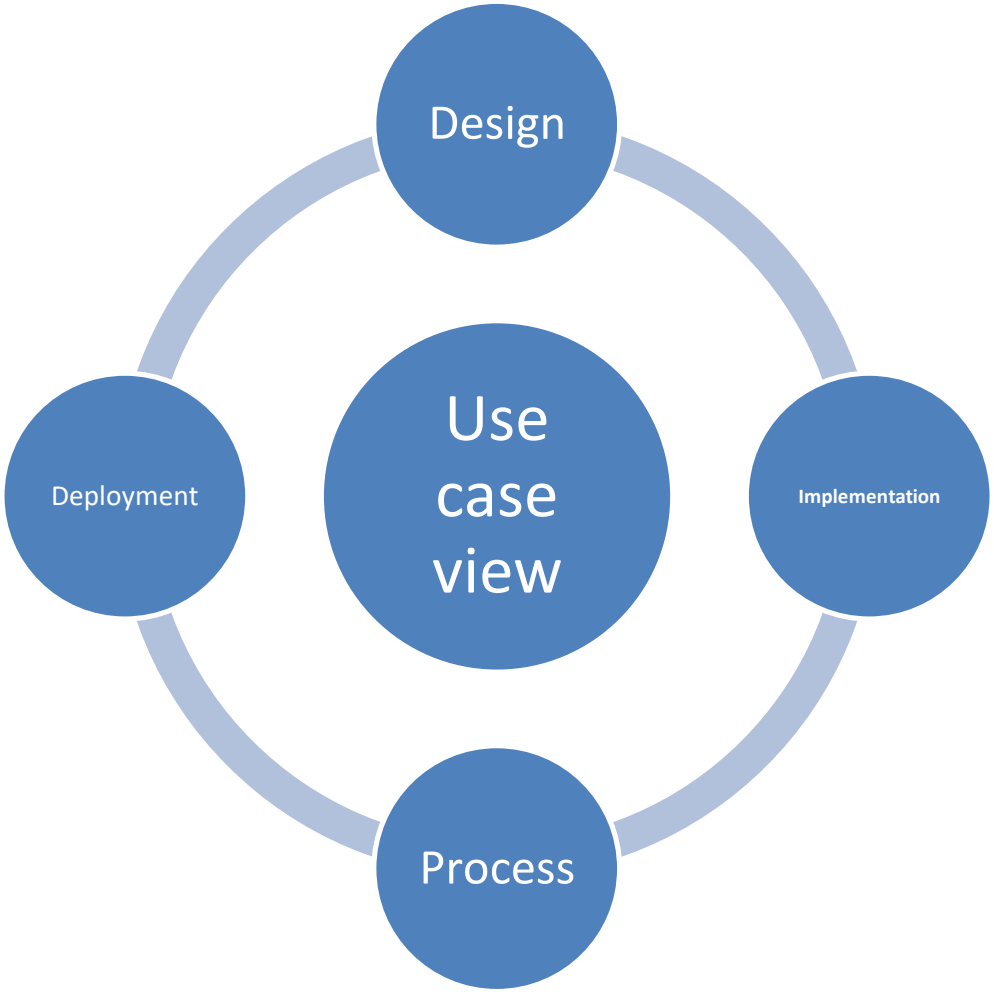


Relationships

- **Realization**- Realization can be defined as a relationship in which two elements are connected.
- One element describes some responsibility, which is not implemented and the other one implements them.
- This relationship exists in case of interfaces.

Realization

Perspectives of UML



The center is the Use Case view which connects all these four. A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case.

Design of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

Classification	Types	Features
Structure Diagrams	Class Diagram	Structure of each class; relationships between classes
	Component Diagram	Components that make up the software and the dependencies between them
	Deployment Diagram	Physical layout of the system
	Package Diagram	Grouping of model elements such as classes and relationships between groups (packages)
Behavioral Diagrams	Use Case Diagram	Functions provided by the system, and relationships with external users and other systems
	Sequence Diagram	Interaction of objects along the time axis
	Collaboration Diagram	Objects interacting to implement some behavior within a context
	Statechart Diagram	Model life time of an object from creation to termination
	Activity Diagram	System operation flow

UML tool links

Rational Rose

http://www-3.ibm.com/software/awdtools/developer/rosexde/

Together

http://www.borland.com/together/

ArgoUML

http://argouml.tigris.org

Visio

Hard to find info on Microsoft's site!

http://msdn.microsoft.com/office/understanding/visio/

Dia

http://www.lysator.liu.se/~alla/dia

References

https://www.tutorialspoint.com/uml/