Unit 1

Unified Modeling Language

Prof. Kirtankumar Rathod
Dept. of Computer Science
Indus University

Unified Modeling Language (**UML**)

- The **Unified Modeling Language** (**UML**) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

- It was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994–1995

UML is…

- Used to visualize, specify, construct, and document
- …*appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems.*
- Process independent.

UML is a Language

- Vocabulary and rules for communication.
- Focus on conceptual and physical representations of a system.
- A language for software blueprints.
- *Not* a process.

UML is for Visualizing

- An explicit model facilitates communication.
- UML is a graphical language.
- UML symbols are based on well-defined semantics.

UML is for Specifying

- Specifying means building models that are:
  - Precise
  - Unambiguous
  - Complete
- UML addresses the specification of all important analysis, design, and implementation decisions.

UML is for Constructing

- Models are related to OO programming languages.
- Round-trip engineering
  - Forward engineering—direct mapping of a UML model into code.
  - Reverse engineering—reconstruction of a UML model from an implementation.
  - Requires tool and human intervention to avoid information loss.

UML is for Documenting

- Architecture
- Requirements
- Tests
- Activities
  - Project planning
  - Release management

Conceptual Model of UML

- Three basic building blocks of UML
  - Things—abstractions that are first class citizens in a model.
  - Relationships—tie things together.
  - Diagrams—group interesting collections of things.

Things

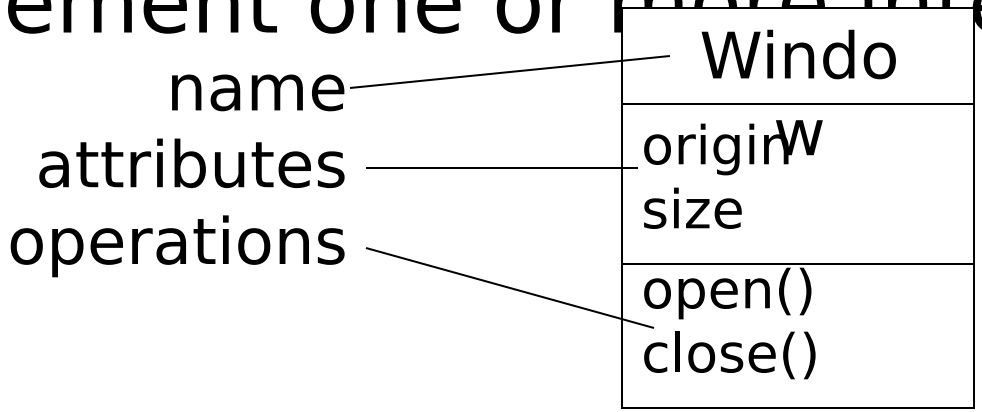
1) Structural—nouns of UML models.
2) Behavioral—dynamic parts of UML models.
3) Grouping—organizational parts of UML models.
4) Annotational—explanatory parts of UML models.

Structural Things

- Nouns of UML models.
- Conceptual or physical elements.
- Seven Kinds
  - Classes
  - Interfaces
  - Collaborations
  - Use cases
  - Active classes
  - Components
  - Nodes

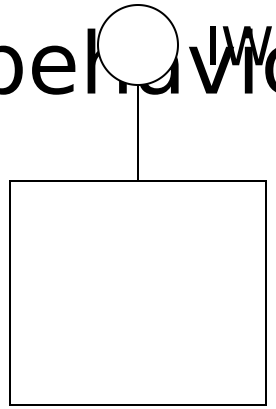
Classes

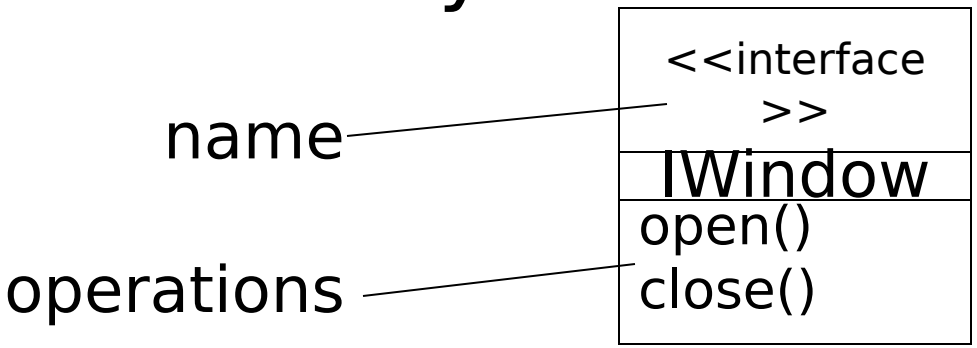
- Description of a set of objects that share the same attributes, operations, relationships, and semantics.
- Implement one or more interfaces.



Interfaces

- Collection of operations that specify a service of a class or component.
- Describes the externally visible behavior.



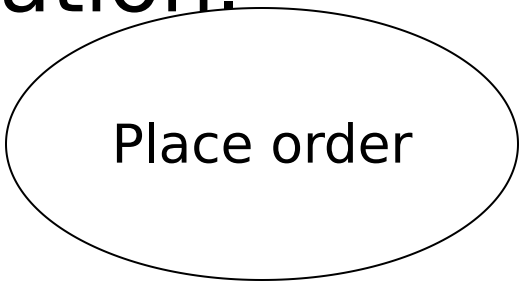


Collaborations

- Defines an interaction.
- Society of roles and other elements.
- Provide cooperative behavior.
- Structural and behavioral dimensions.

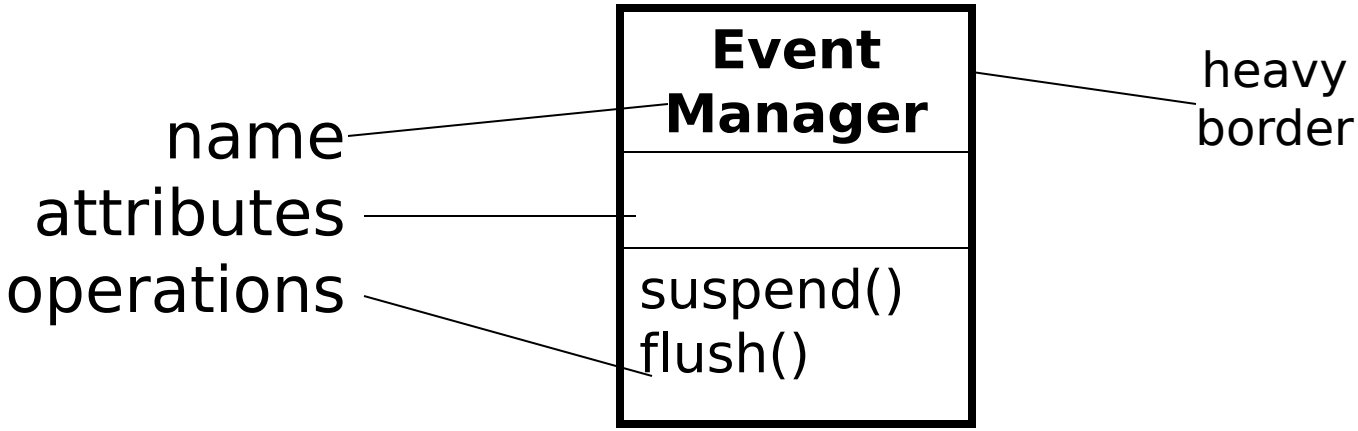
Use Cases

- Description of a sequence of actions that produce an observable result for a specific actor.
- Provides a structure for behavioral things.
- Realized by a collaboration.



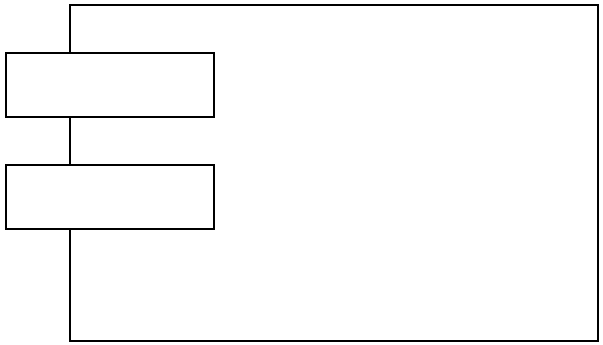
Active Classes

- Special class whose objects own one or more processes or threads.
- Can initiate control activity.



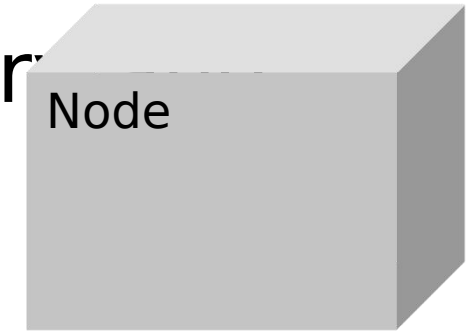
Components

- Physical and replaceable part.
- Conforms to a set of interfaces.
- Physical packaging of logical components.



Node

- Physical element that exists at run time.
- Represents a computational resource.
- Generally has memor
  processing power.



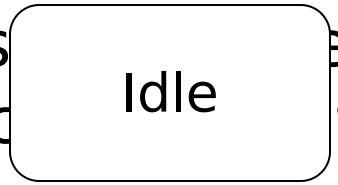
Variations on Structural Things

- Actors
- Signals
- Utilities
- Processes and Threads
- Applications
- Documents
- etc.

Behavioral Things

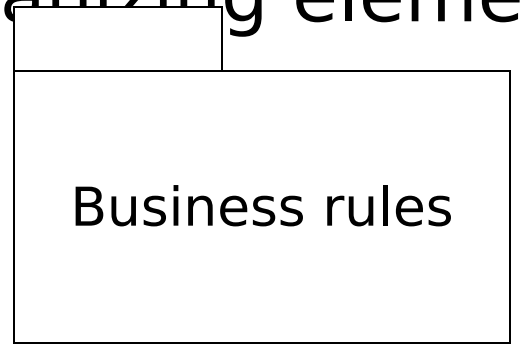
- Verbs of UML models.
- Dynamic parts of UML models.
- Usually connected to structural elements.
- Two kinds
  - Interactions—behavior that comprises a set of messages exchanged among a set of objects.

  - State Machines—specifies ences of states an object or interac through in response to events.



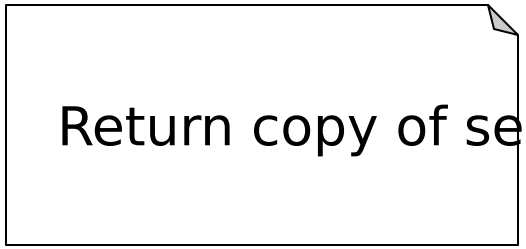
Grouping Things

- Organizational parts of UML.
- Purely conceptual; only exists at development time.
- One kind
  - Package—general-purpose mechanism for organizing elements.



Annotational Things

- Explanatory parts of UML.
- Comments regarding other UML elements.
- Information best expressed as text.
- One kind
  - Note—symbol for rendering constraints or comments attached to an element.

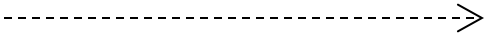


Relationships in UML

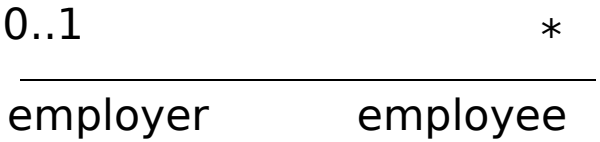
- Dependency
- Association
- Generalization
- Realization

Relationships

- Dependency—semantic relationship between two things in which a change to one thing may affect the semantics of the other.



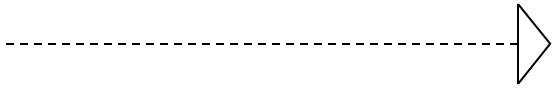
- Association—structural relationship that describes a set of links; a link being a connection among objects.



Relationships (cont’d)

- Generalization—specialization relationship in which child objects are substitutable for the parent.

- Realization—semantic relationship between classifiers, wherein one classifier specifies a contract that the other guarantees to carry out.



Diagrams

- Graphical representation of a set of elements.
- Rendered as a connected graph
  - Vertices are things.
  - Arcs are behaviors.
- Projection into a system form a specific perspective.
- Five most common views built from nine diagram types.

Common Diagram Types

- Class
- Object
- Use case
- Sequence
- Collaboration
- Statechart
- Activity
- Component
- Deployment

Rules of UML

- Well formed models—*semantically self-consistent and in harmony with all its related models*.
- Semantic rules for:
  - Names—what you can call things.
  - Scope—context that gives meaning to a name.
  - Visibility—how names can be seen and used.
  - Integrity—how things properly and consistently relate to one another.
  - Execution—what it means to run or simulate a dynamic model.

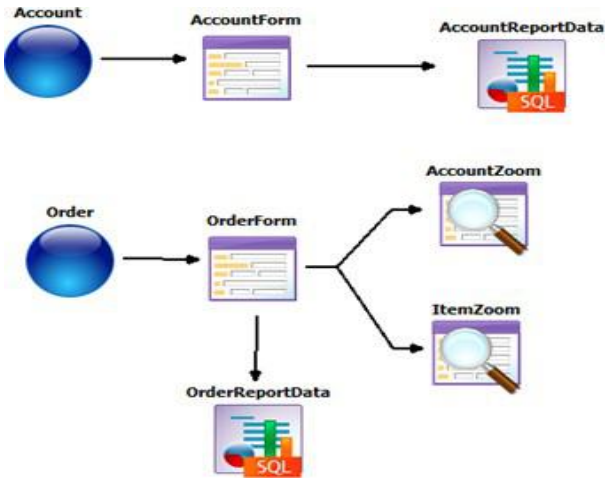
Object-Oriented Modeling and Design with UML

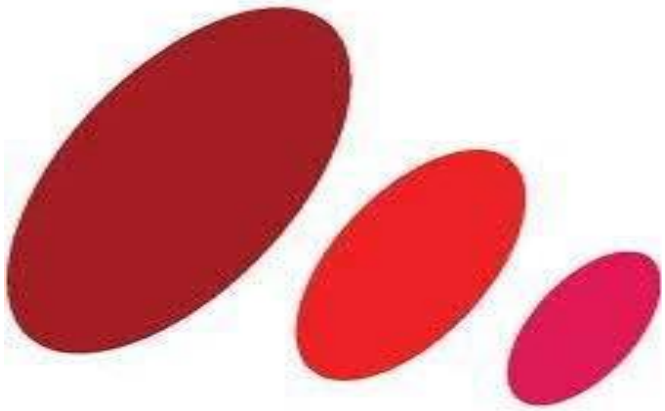
Contents

- **Introduction**
- **OO characteristics**
- **OO development**
- **OO themes**
- **Summary**

Introduction

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





OO characteristics (1

Identity

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

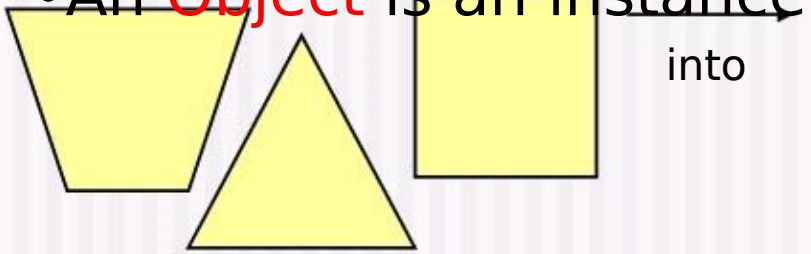
- Each object has is own inherent identity
  - Two objects are **distinct** even if all their attribute values are **identical**

- 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

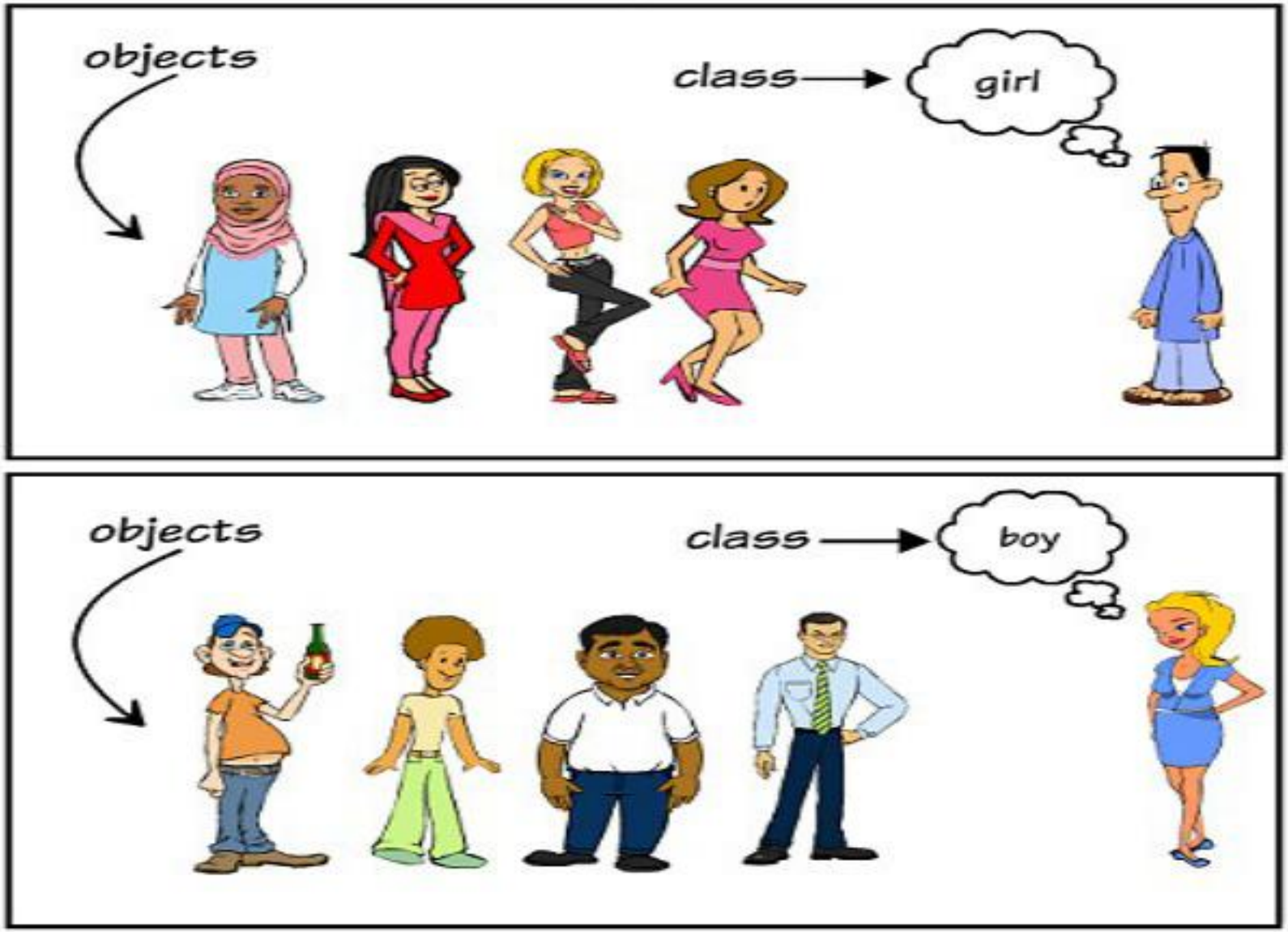
abstract
into

Bicycle class

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

Operations
- shift
- move
- repair

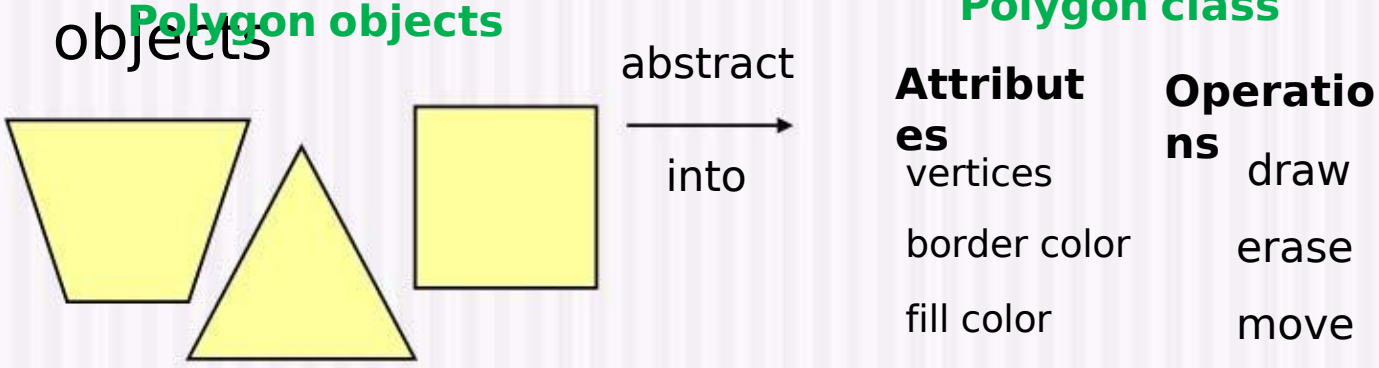
Examples for Class &



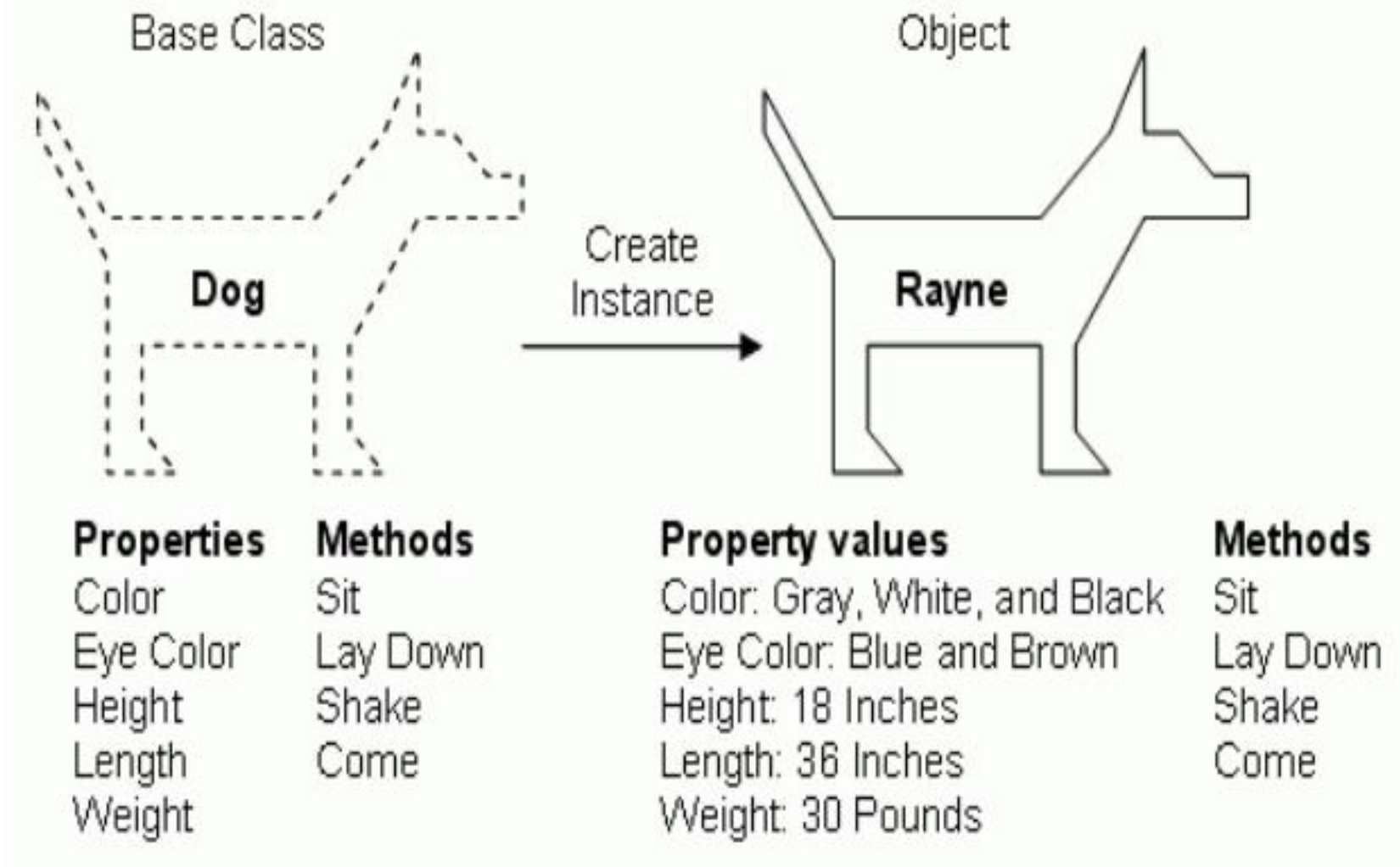
OO characteristics

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
- An Object is an instance of a class
- Each class describes an infinite set of individual objects



An Object is an instance of a class

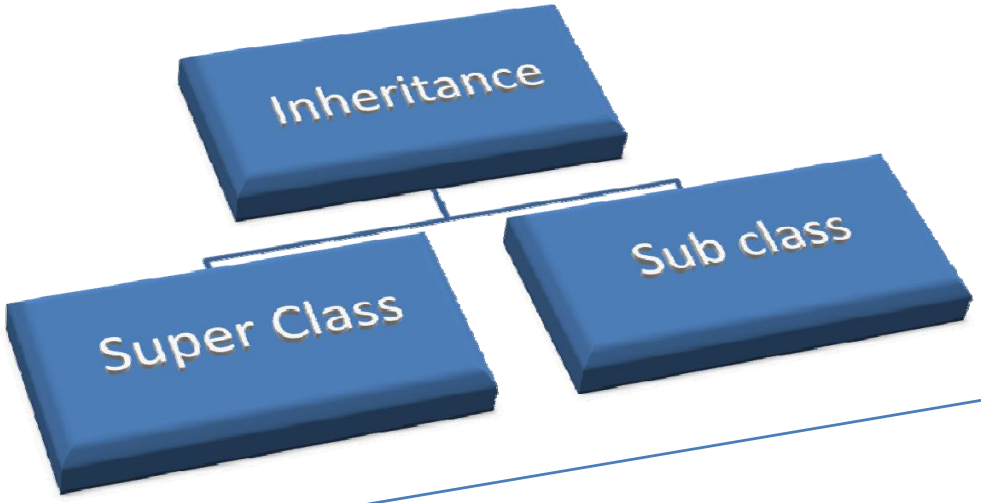


Derive the instances for the c

Class		Object	
Attributes	Operations/Methods	Attributes Values	Operations/Methods
?		?	

OO characteristics(3/4)

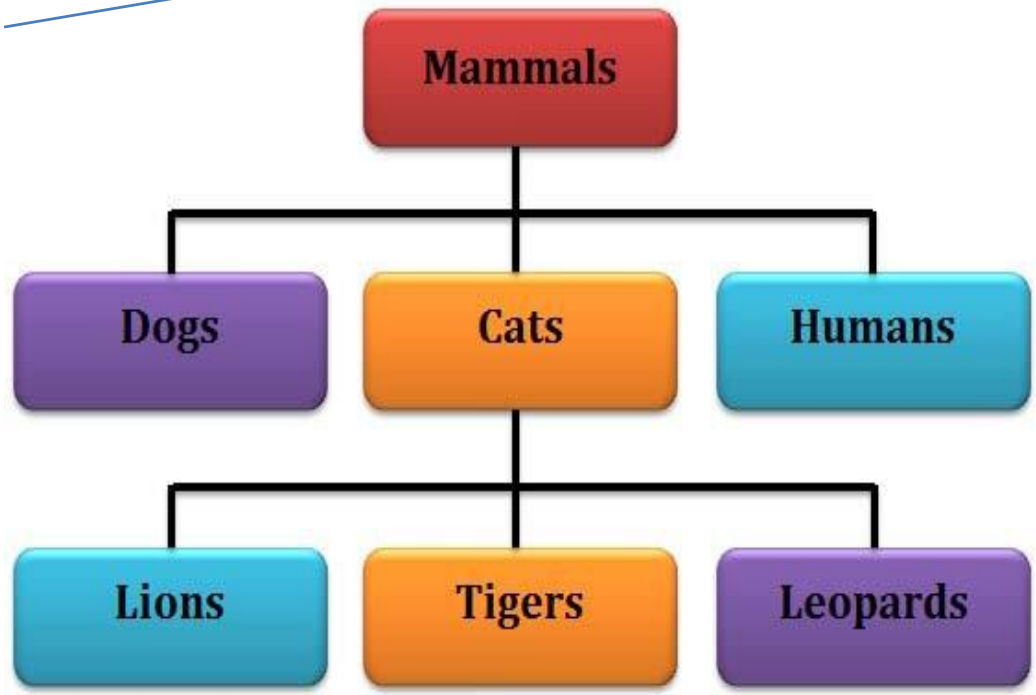
Inheritance



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



The simple hierarchy of Mammals

OO characteristics (4/4)

Polymorphism

- **Same operation** may behave **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

OO development (1/6)

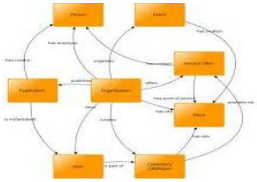
- **Essence of OO development**
  - Identification and organization of application concepts rather than their final representation in **programming language.**

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 proces** ndent of a ning language (OO is fun lly a way and

OO development (2/6)

- **OO Methodology**
  - Process for OO development with graphical notation (OO Concepts)
  - Methodology = building a model + adding details during design
  - Same notation is used fr
    - 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

System conception (1/5)

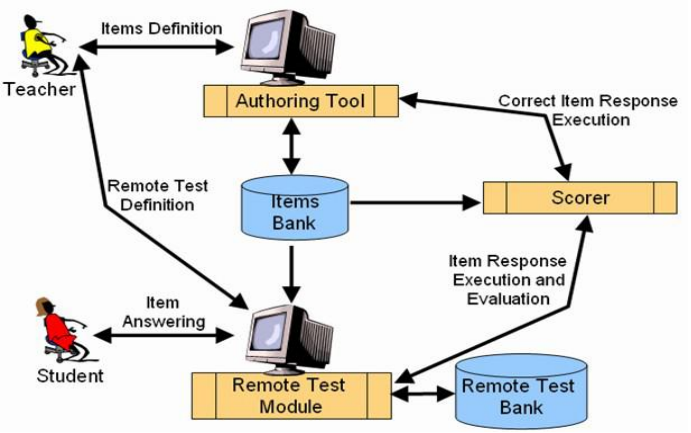
s/w development begins with **business analyst** and formulate **tentative requirements**

Analysis(2/5)

- Restates the requirements from system conception by constructing model
- **statements** must work with the **requestor** to understand the **problem**
- Analysis model (abstract) describes **what to 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)



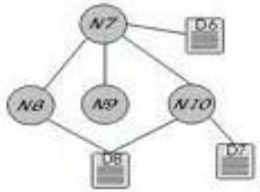
System Design(3/5)

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

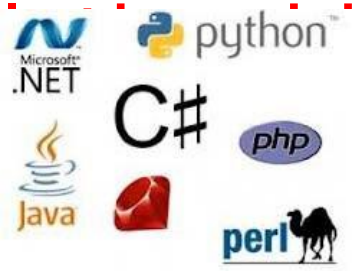
- **System designer decides**
  - what performance characteristics to c
  - Choose a strategy of attacking the **pr**
  - Make tentative resource allocation.

Methodology Stages Class Design (4/5)

- 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 **da structure** and **algorithm**.



Methodology Stages Implementation

- Implement translate class and relationship


Programming language, DB, H/W

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


- Follow good software engineering

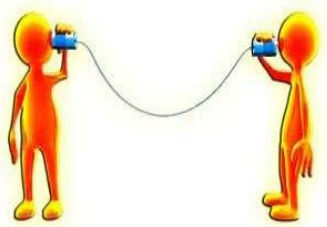
OO development (3/8)

Modeling

A model is a simplification of reality

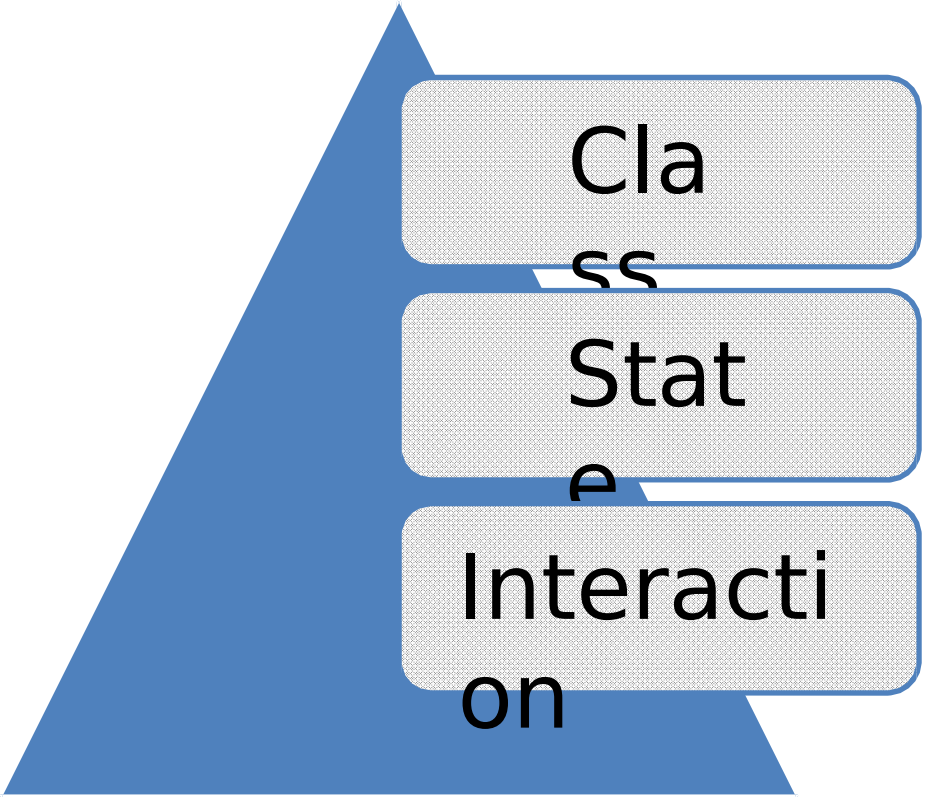
- **Abstraction** for the purpose of understanding before building it
- **Isolate** those aspects which are important and **suppress** the rest(unimportant)

Purpose



- Testing a physical entity before bu
- Communication with custom
- Visualization
- Reduction of complexity

Design a System



For the objects in the system and their relationship

For the life history of the object

For the interaction among the objects

OO development (4/6)

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
  - Provides context for **state** and **interaction model**
- Goal
  - Capture **important concepts** of an application from the real world
- Representation
  - Class diagrams → Graph
    Nodes: Class
    Arc: relationship B/W Classes
  - Generalization, aggregation

OO development (5/6)

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 :

OO development (6/6)

Three models (Cont'd)

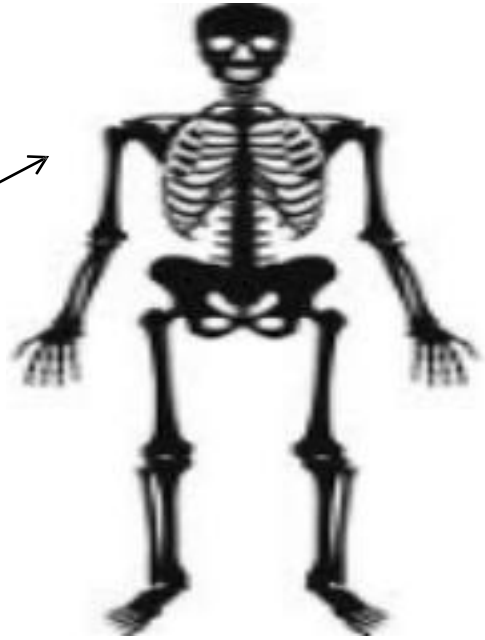
Interaction model

- 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 (1/6)

Abstraction

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



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

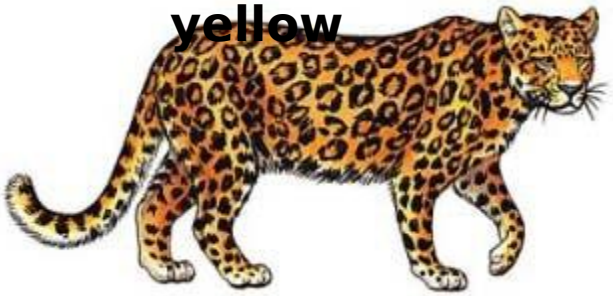
- What an object is and does, before deciding how to implement

- 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**



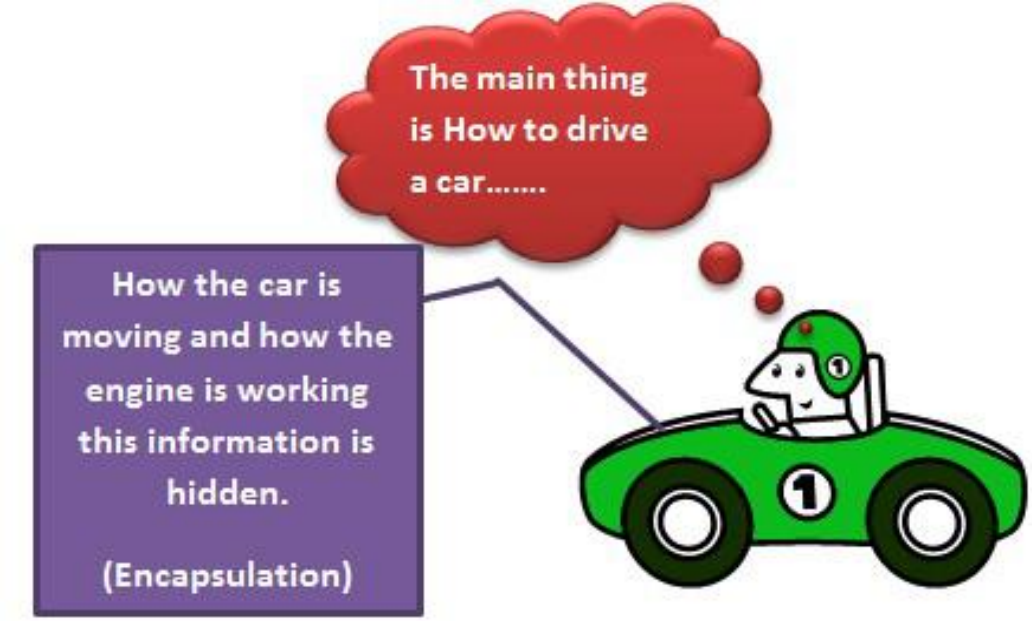
running is very **fast**

color is **black**

running is very **slow**

OO themes (2/6)

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

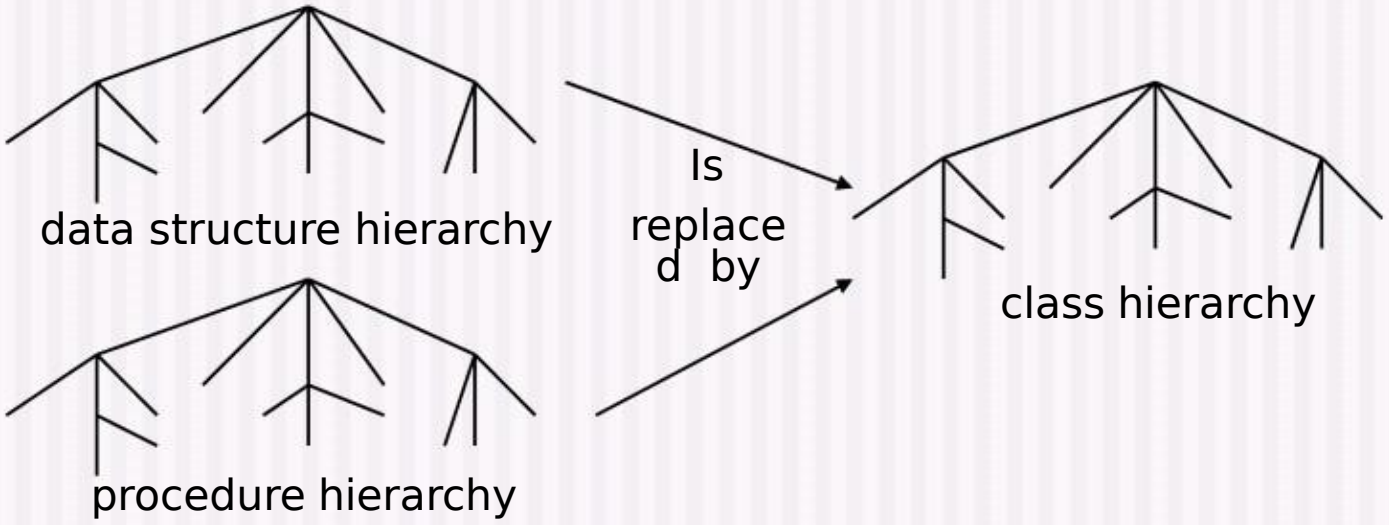
Example

- 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 (3/6)

Combining data and behavior

Data structure hierarchy matches the operation inheritance hierarchy



Old approach

OO approach

oo themes

Sharing (4/6)

No redundancy (Inheritance)

Reusability (tools- abstraction, inheritance, encapsulation)

Emphasis on the essence of an object (5/6)

Focus on **what an object is**

- Rather than how it is used

Synergy (6/6)

Identity, classification, polymorphism, inheritance

- Be clearer, more general and robust

Unified Modeling Language User Guide

Prof. Kirtankumar Rathod
Dept. of Computer Science
Indus University

What is a model?

- *A model is a simplification of reality.*
- A set of blueprints of a system.
- *Semantically closed abstraction of the system.*
  - *A model is an abstraction of something for the purpose of understanding it before building it.*

Why We Model

- **Communicate** a desired structure and behavior of a software system.
- **Visualize and control** a system's architecture.
- **Assist** in understanding a system under development.
- **Expose** opportunities for simplification and reuse.
- **Manage** risk.
- **Document** decisions.

Principles of Modeling

1. The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.

2. Every model may be expressed at different levels of precision.

Principles of Modeling

3. The best models are connected to reality.

4. No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.