THE OBJECT MODEL

- Programming Language Generations
- What is an Object?
- Programming Paradigms
- Elements of the Object Model
- Relationships Among Objects
- Classification

PROGRAMMING LANGUAGE GENERATIONS

- New Focus: programming in the large
- High-order languages dominate

Ada, C++

1962-1970: 3rd Generation

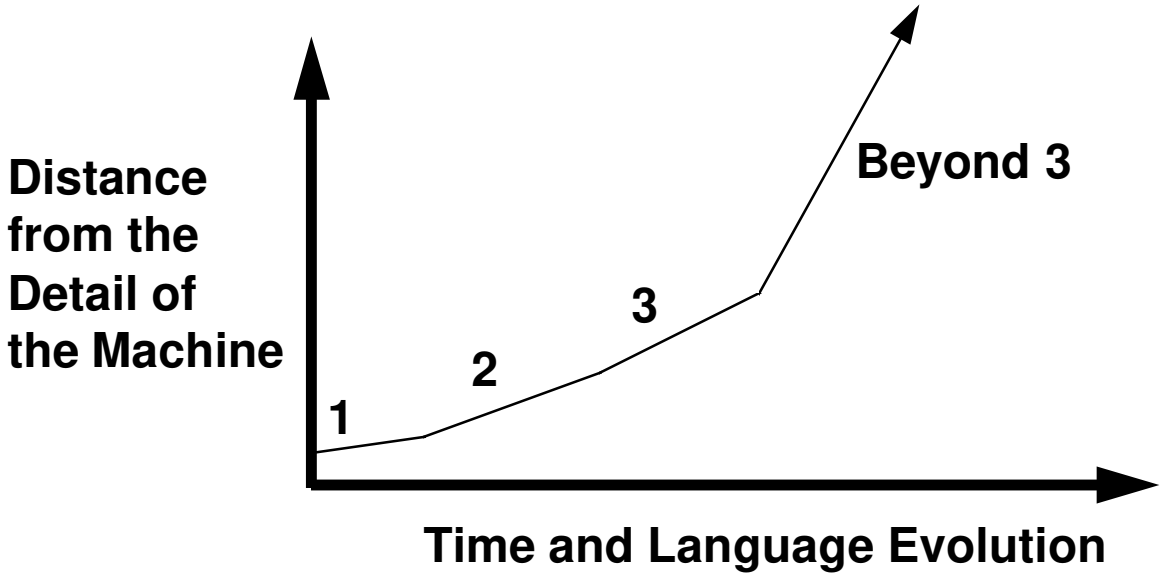
1959-1961: 2nd Generation

1954-1958: 1st Generation

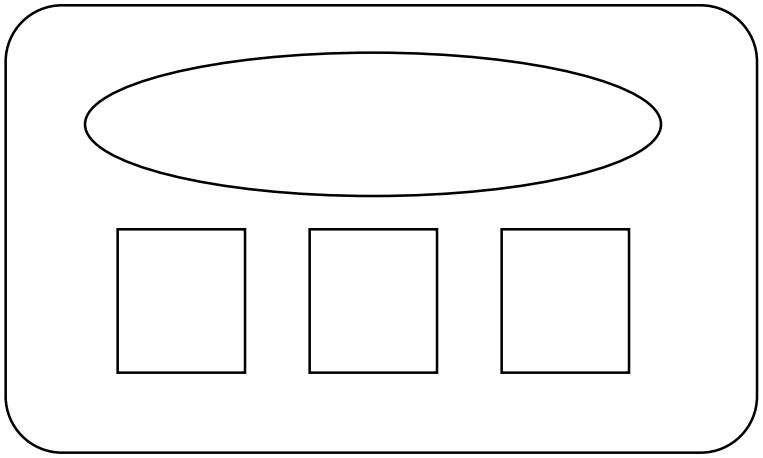
Gen	*Sample Languages*	*Language Features*
1	FORTRAN I	Mathematical expressions
2	FORTRAN II, COBOL	Subroutines, data handling
3	Pascal, Simula	Blocks, typing, classes

Evolution of Abstraction

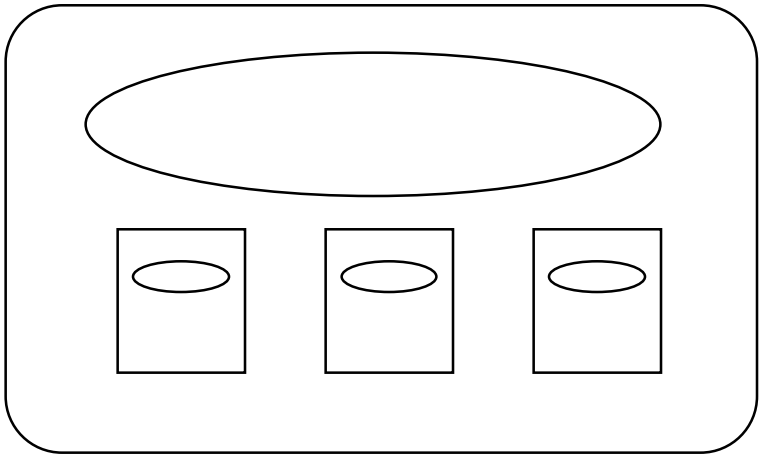
Gen	*Kind of Abstraction*
1	Mathematics
2	Algorithm and procedures
3	Data and data models of real-world entities
Beyond	Objects and object models of real-world entities



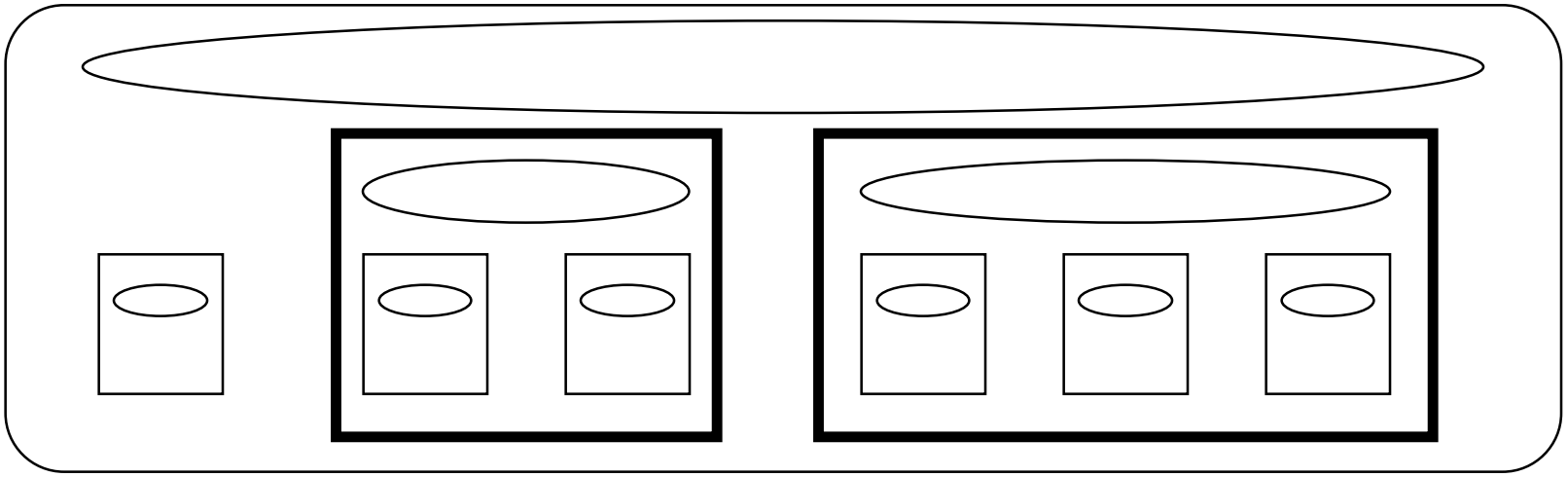
Topologies of Languages by Generation



1st Generation



2nd Generation



3rd Generation and Beyond

A Shift in Focus

Given that:

Verbs => Procedures and Functions

Nouns => Data

Then:

Function-oriented Program = Collection of Verbs Supported by Nouns

Object-oriented Program = Collection of Nouns Supported by Verbs

Which is a More Realistic Model of the World?

- **A collection of functions being performed?**
- **A collection of objects interacting with each other?**

WHAT IS AN OBJECT?

Informal, Intuitive Definition

An *object* is an integral entity which can:

- **change state**
- **behave in certain discernable ways**
- **be manipulated by various forms of stimuli**
- **stand in relation to other objects**

***Objects* :**

- **exist, occupy space, and assume a state**
- **possess attributes**
- **exhibit behaviors**

Formal Definition

Object Concept **- objects have a permanence and identity apart from any operation upon them**

Formal definition of an object from the perspective of OOD:

Object **- an entity which has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 77

Examples of Non-Objects

- Attributes, such as time, beauty, or color
- Emotions, such as love or anger
- Entities which are normally objects but are, instead, thought of as attributes of objects when a particular problem space is considered

Temperature

```
Oven_Temp : TEMPERATURE
   := 350.0;  -- degrees F
```

Temperature as an object

Temperature Sensor

```
type SENSOR is record
   Temp : TEMPERATURE;
   Redundancy : MULTIPLEX;
   Location: MEMORY_ADDRESS;
end record;
Oven_Temp : SENSOR := (
   Temp => 350.0,  -- degrees F
   Redundancy => TRIPLEX,
   Location => 16#1a0#  );
```

Temperature as an attribute of an object

Kinds of Software Objects

- Real-world, tangible objects with boundaries that may or may not be clearly defined
- Inventions of the design process which collaborate with other objects to provide some higher-level behavior
- Intangible events or processes with well-defined conceptual boundaries

Tangible	Clearly-defined boundaries No clearly-defined boundaries
Intangible	Clearly-defined boundaries No clearly-defined boundaries Events or processes Inventions of the design process

Dissecting an Object

Formal definition of an object from the perspective of OOD:

> ***Object*** **- an entity which has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable**
>
> -- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 77

This definition of an object refers to three key features:

- **State**
- **Behavior**
- **Identity**

These key features will be discussed in detail.

State

State of an object **- encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 78

Property or attribute of an object **- a part of the state of the object which is an inherent or distinctive characteristic, trait, quality, or feature that contributes to making an object uniquely that object**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 78

State of an Object - Example

Temperature Sensor

```
type TEMPERATURE_SENSOR is record
   Temp : TEMPERATURE;  -- degrees F
   Redundancy : MULTIPLEX;
   Location: MEMORY_ADDRESS;
end record;
Oven_Temp : TEMPERATURE_SENSOR := (
   Temp => 350.0,  -- degrees F
   Redundancy => TRIPLEX,
   Location => 16#1a0#  );
```

Objects of class TEMPERATURE_SENSOR, such as Oven_Temp, have three attributes:

- ***Temp*, a dynamic attribute which changes with time**
- ***Redundancy*, a static attribute (the number of sensed points) which is fixed when the object is created**
- ***Location*, a static attribute which is fixed when the object is created**

Behavior

Behavior of an object **- how an object acts and reacts, in terms of its state changes and message passing**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 80

Operation **-- some action that one object performs upon another in order to elicit a reaction**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 80

The terms *operation* and *message* are interchangeable.

Method **-- operation that a client may perform upon an object**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 80

Behavior of an Object - Example

```
package Temperature_Sensor is

  type STATUS is (NOT_OK, OK);
  type TEMPERATURE is FLOAT range -400.0 .. 3_000.0; -- deg F
  type MULTIPLEX is (SIMPLEX, DUPLEX, TRIPLEX);
  type MEMORY_ADDRESS is INTEGER range 0 .. 1_024;

  type OBJECT is record
    Temp       : TEMPERATURE;
    Redundancy : MULTIPLEX;
    Location   : MEMORY_ADDRESS;
  end record;

  function Current_Temperature (Item : in OBJECT)
    return TEMPERATURE;

  function Reliability (Item : in OBJECT)
    return STATUS;

end Temperature_Sensor;
```

Behavior - Kinds of Operations

- ***Modifier*** **- an operation that alters the state of an object, such as a get_with_update or put operation**
- ***Selector*** **- an operation that accesses the state of an object, but does not alter the state, such as a get operation**
- ***Iterator*** **- an operation that permits all parts of an object to be accessed in some well-defined order, such as movement through a linked list**
- ***Constructor*** **- an operation that creates and object and/or initializes its state**
- ***Destructor*** **- an operation that frees the state of an object and/or destroys the object itself**

Behavior - The Protocol of an Object

Protocol **- all of the methods and free subprograms [procedures or functions that serve as nonprimitive operations upon an object or objects of the same or different classes] associated with a particular object**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Pp 82-83

The protocol of an object defines the envelope of that object's allowable behavior, comprising the entire external view of the object (both static and dynamic).

Behavior - Objects as Machines

The *Finite State Machine* provides a good model for some objects.

Objects may be either active or passive:

- ***Active Object*** **- an object that encompasses its own thread of control**
- ***Passive Object*** **- an object that does not encompass its thread of control**

Identity

Identity **- that property of an object which distinguishes it from all other objects**

-- Khoshafian and Copeland, "Object Identity," *SIGPLAN Notices*, Volume 21, Issues 11, November 1986, Page 406

I : Integer range 1..20 := 12;

identity of the object

Identity - Object Assignment

Object Assignment **differs from copying in that in object assignment, the identity of an object is duplicated by assignment to a second name. Two names then refer to the same object.**

Conventional Assignment **refers to the act of copying the state information of one object into another object. The state of two objects is now the same, but the state of one object may be changed without affecting the other.**

Identity - Equality

Like assignment, *Equality* can have two meanings:

- **two names are equal if they designate the same object**
- **two names are equal if they designate different objects but their state is identical**

What is a Class?

Class **- a set of objects that share a common structure and a common behavior**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 93

A class represents only an *abstraction*, whereas an object, *an instance of a class*, is a concrete entity that exists in time and space.

The Class as a Contractual Binding

The class captures the structure and behavior common to all related objects, serving as a binding contract between an abstraction and all of its clients.

Strongly typed programming languages can detect violations of the contract that is a class during compilation.

Two views of a class:

- ***Interface*** **- the outside view of a class, emphasizing the abstraction while hiding the structure and details of how its behavior works**
- ***Implementation*** **- the inside view of a class, which details the internal structure of a class and the details of how its behavior works**

The Interface to a Class

The interface to a class consists of:

- **primarily, the declarations of all operations applicable to instances of the class; these operations may be invoked by clients of the class objects**
- **the declaration of other classes**
- **constants**
- **variables**
- **exceptions**

The last four are included if they are needed to complete the abstraction.

The Interface to a Class, Continued

The interface to a class can be divided into three parts:

- ***Public*** **- a declaration that is visible to all clients of the objects of a class**
- ***Protected*** **- a declaration that is not visible to any other classes except the subclasses of the class**
- ***Private*** **- a declaration that is not visible to any other classes**

C++ does the best job in allowing a developer to make explicit distinctions among these different parts of a class interface. Ada permits declarations to be public or private, but not protected.

The State of an Object

- Defined in private part of class declaration
- Constant and variable declarations

Why is the State of an Object NOT in the Implementation?

- Needed by the compiler
- Technology not sufficiently advanced

PROGRAMMING PARADIGMS

Most programmers work in one language and use only one programming style. They program in a paradigm enforced by the language they use. Frequently, they have not been exposed to alternate ways of thinking about a problem, and hence have difficulty in seeing the advantage of choosing a style more appropriate to the problem at hand.

-- Jenkins and Glasgow, "Programming Styles in Nail," *IEEE Software*, Volume 3, Number 1, Page 48 (Jan 1986)

Main Kinds of Programming Paradigms

Paradigm	*Kinds of Abstractions Employed*
Procedure-oriented	**Algorithms**
Object-oriented	**Classes and objects**
Logic-oriented	**Goals, often expressed in a predicate calculus**
Rule-oriented	**If-then rules**
Constraint-oriented	**Invariant relationships**

ELEMENTS OF THE OBJECT MODEL

The *Object Model* is the conceptual framework for all things object-oriented.

Major Elements

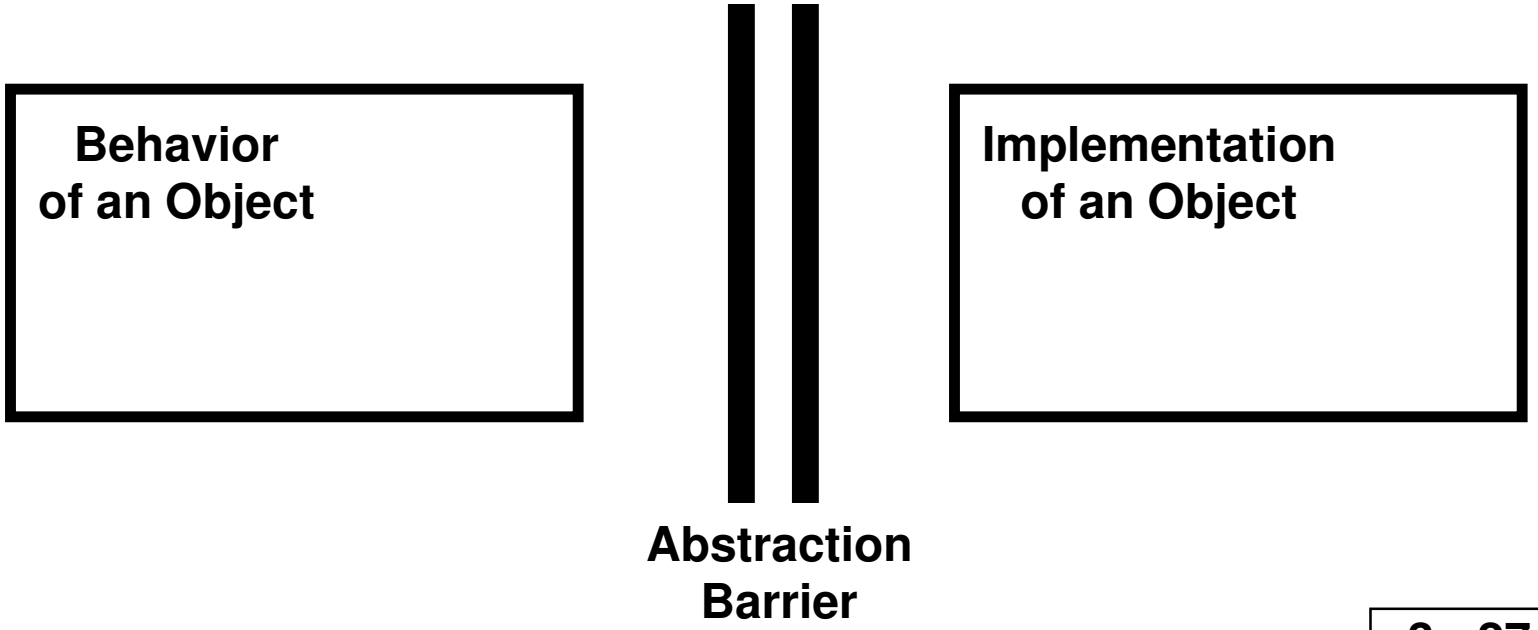
- ✓ **Abstraction**
- ✓ **Encapsulation**
- ✓ **Modularity**
- ✓ **Hierarchy**

Minor Elements

- ✓ **Typing**
- ✓ **Concurrency**
- ✓ **Persistence**

Abstraction

Abstraction - what distinguishes one kind of object from another kind of object



Abstraction and the Problem Domain

Deciding on the correct set of abstractions for a given problem domain is the central problem in object-oriented design.

"Determining the correct set of abstractions" is covered in detail in the next Module.

Kinds of Abstraction

Entity abstraction **- an object that represents a useful model of an entity in the problem domain**

Action abstraction **- an object that provides a generalized set of operations, all of which perform the same kind of function**

Virtual machine abstraction **- an object that groups together operations that are all used by some superior level of control or operations that all use some junior-level set of operations**

Coincidental abstraction **- an object that packages a set of operations that have no relation to each other**

Entity Abstractions

Client **- an object that uses the resources of another object**

Behavior of an object **- the operations that a client may perform upon the object (the *protocol* of the object) and the operations that the object may perform upon other objects**

All entity abstractions may have two kinds of properties:

- ***Static*** **- fixed for the life of the object; example: a file's name or identity**
- ***Dynamic*** **- can vary during the life of the object; example: a file's size**

Encapsulation

Encapsulation, or Information Hiding **- the process of hiding all the details of an object that do not contribute to its essential characteristics**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 46

Modularity

Modularity **- the property of a system that has been decomposed into a set of cohesive and loosely coupled modules**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 52

Issues Concerning Modularity

Technical --

- **Class and object selection - modules are the containers of the classes and objects**
- **Logically-related classes and objects grouping**
- **Visibility of modules to other modules**
- **Isolation of system dependencies**
- **Reuse of modules across applications**
- **Limits placed on the size of object code segments, particularly when a compiler places one and only one module into one and only one object code segment**

Non-Technical --

- **Work assignments may be given on a module basis**
- **Modules usually serve as configuration items**
- **Some modules may require more security**

Modules and Classes/Objects

Two entirely independent design decisions:

- **Finding the right classes and objects**
- **Organizing the classes and objects into separate modules**

Hierarchy

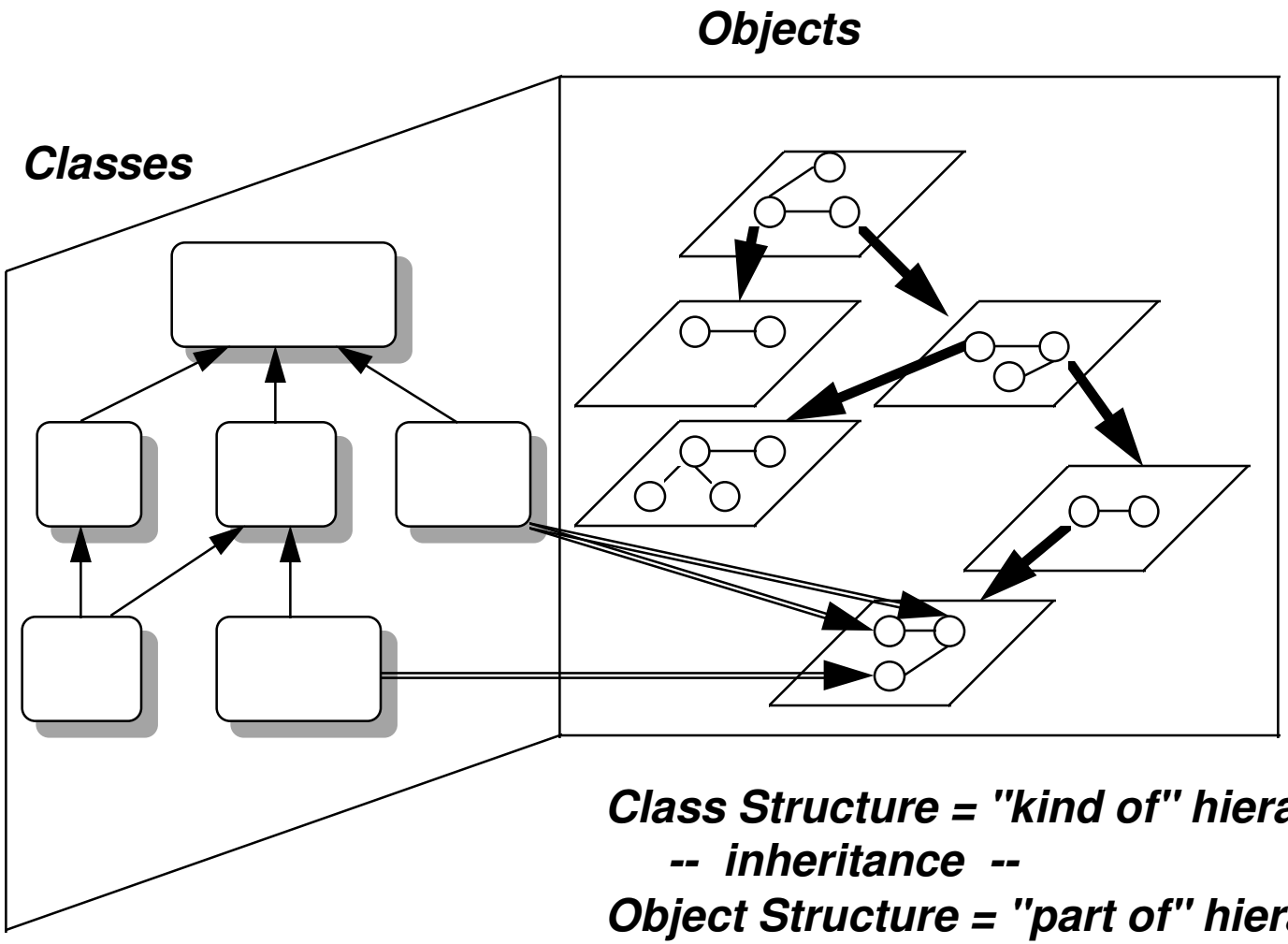
***Hierarchy* - the ranking or ordering of abstractions**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 54

The two most important hierarchies in a complex system:

- **the *class structure* (the "kind of" hierarchy)**
- **the *object structure* (the "part of" hierarchy)**

Two Key Hierarchies



Class Structure = "kind of" hierarchy
-- inheritance --
Object Structure = "part of" hierarchy
-- aggregation --

Typing

Typing **- the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 59

Some Benefits of Strong Typing

- Runtime crashes of programs reduced
- Early error detection
- Type declarations help to document programs:

```
type VELOCITY is new FLOAT range 0.0 .. 1_000.0; -- MPH
X : VELOCITY;
Y : FLOAT;
```

- Code efficiency may be improved

Static Typing and Dynamic Binding

Static Typing, Static Binding, or Early Binding **- the types of variables are fixed at compile time**

Dynamic Binding or Late Binding **- the types of variables are not known until runtime**

Polymorphism and Typing

Polymorphism **- the concept in type theory in which a single name (such as a variable declaration) may denote objects of many different classes that are related by some common superclass**

Monomorphism **is the opposite of polymorphism, so a monomorphic object may only respond to the set of operations associated with its own class.**

Concurrency

Concurrency **- the property that distinguishes an active object from one that is not active**

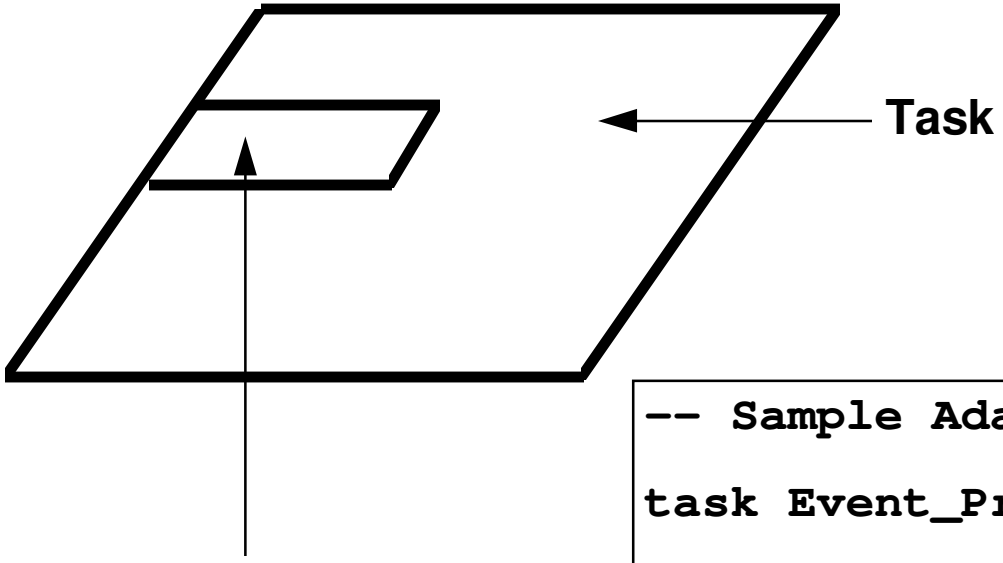
-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 66

An object is an excellent candidate for a concurrent entity because:

- **it implicitly defines a unit of distribution and activity**
- **it explicitly defines a communication interface**

Tasks as Concurrent Objects

In Ada, the Ada runtime system implements the tasking model. This model can be implemented on one or many CPUs.




```
-- Sample Ada Task Specification
task Event_Process is
  entry Trigger (Input : in KIND);
end Event_Process;
-- Creating two Event_Process tasks
Processor1, Processor2 : Event_Process;
```

Persistence

Persistence **- the property of an object through which its existence transcends time (i.e., the object continues to exist after its creator ceases to exist) and/or space (i.e., the object's location moves from the address space in which it was created)**

-- Grady Booch, *Object-Oriented Design with Applications*, 1991, Page 70

RELATIONSHIPS AMONG OBJECTS

An object of and by itself is usually uninteresting. However, a system of objects, wherein the objects collaborate with one another to define the behavior of the system, is intensely interesting.

Two kinds of object hierarchies are extensively employed in OOD:

- **Using relationships, where one object employs the resources of another**
- **Containing relationships, where one object contains one or more other objects**

Using Relationships

Three roles:

- ***Actor*** **- operates upon other objects; an *active object***
- ***Server*** **- is only operated upon by other objects; a *passive object***
- ***Agent*** **- can do both**

Using Relationships, Continued

The need for synchronization in an environment involving multiple threads of control leads to another way to classify kinds of objects:

- ***Sequential object*** **- a passive object whose semantics are guaranteed only in the presence of a single thread of control**
- ***Blocking object*** **- a passive object whose semantics are guaranteed in the presence of multiple threads of control**
- ***Concurrent object*** **- an active object whose semantics are guaranteed in the presence of multiple threads of control**

Containing Relationships

In a *containing relationship*, an object may encapsulate one or more other objects.

Advantages:

- **Reduce the number of objects**

Disadvantages:

- **Sometimes leads to undesirable tighter coupling**

CLASSIFICATION

- What is Classification?
- Classification and OOD
- Why is Classification So Hard?
- Approaches to Classification
- Domain Analysis

What is Classification?

Classification **- the means whereby we order knowledge**

Recognizing the "sameness" among things

No single approach

Classification and OOD

The identification of classes and objects is the hardest part of OOD.

The identification of classes and objects involves:

- **discovery, through which we recognize the key abstractions and mechanisms that form the vocabulary of our problem domain**
- **invention, through which we devise generalized abstractions and new mechanisms that regulate how objects should collaborate**

Why is Classification so Hard?

- There is no such thing as a "perfect" classification, although some classifications are better than others.
- Any classification is relative to the perspective of the observer doing the classification.
- Intelligent classification requires a tremendous amount of creative insight.

Why is a LASER beam like a goldfish?
Because neither one can whistle.

Creative insight or idiocy?

How is a speck of dust like a thought?
Both can be conceived of.

Approaches to Classification

- Classical categorization
- Conceptual clustering
- Prototype theory

Classical Categorization

Classical Categorization **- all entities that have a given property or set of properties in common form a category; such properties are necessary and sufficient to define the category**

-- Lakoff, G. *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*, 1987, The University of Chicago Press, Page 161

Conceptual Clustering

Conceptual Clustering **- a modern variation on the classical approach in which classes (clusters of entities) are generated by formulating conceptual descriptions of these classes and then classifying the entities according to the descriptions**

Conceptual clustering is a probabilistic clustering of objects.

Prototype Theory

Prototype Theory **- a class of objects is represented by a prototypical object**

Classification and OOD Revisited

An approach proposed by Grady Booch:

1. **Identify classes and objects according to the properties relevant to the application domain.**
2. **If this fails, cluster objects by concepts.**
3. **If either (1) or (2) fail, classify by association, through which clusters of objects are defined according to how closely each resembles some prototypical object.**

Sources of Classes and Objects

Proposed by Shlaer and Mellor:

- **Tangible things, such as cars, telemetry data, and sensors**
- **Roles, such as mother, teacher, and politician**
- **Events, such as landing, interrupt, and request**
- **Interactions, such as loan, meeting, and intersection**

Proposed by Ross (from the perspective of data modeling):

- **People - humans who carry out some function**
- **Places - areas set aside for people or things**
- **Things - tangible physical objects or groups of objects**
- **Organizations - collections of people, resources, facilities, and capabilities that have a defined mission**
- **Concepts - principles or ideas not tangible used to track activities and/or communications**
- **Events - things that happen**

Sources, Continued

Proposed by Coad and Yourdon:

- **Structure - "kind of" and "part of" relationships**
- **Other systems - external systems with which the application interacts**
- **Devices - devices with which the application interacts**
- **Events remembered - a historical event that must be recorded**
- **Roles played - the different roles users play in interacting with the application**
- **Locations - physical locations, offices, and sites important to the application**
- **Organizational units - groups to which users belong**

Domain Analysis

Domain Analysis **- the process of identifying the classes and objects that are common to all applications within a given domain**

Contrast Domain Analysis to Object-Oriented Analysis, which focuses on one problem at a time.

Suggested Steps in Domain Analysis

- Construct a generic model -- consult with domain experts
- Examine existing systems
- Identify similarities and differences between the systems
- Refine the model