ENCE 688R Civil Information Systems

Working with Objects and Classes

Mark Austin

E-mail: austin@isr.umd.edu

Department of Civil and Environmental Engineering, University of Maryland, College Park

Topics: Working with Objects and Classes

Part 1: Motivation and Approach

- Limitations in Functional Approaches to Development.

Part 2: Working with Objects

- Big picture ideas behind object-based modeling.

Part 3: Object Modeling Techniques

- Objects and classes, association relationships, encapsulation, data hiding.
- Inheritance mechanisms, aggregation/composition.

Part 4: Applications

- Systems development framework for multiple stakeholders.
- Points, lines and regions for GIS.

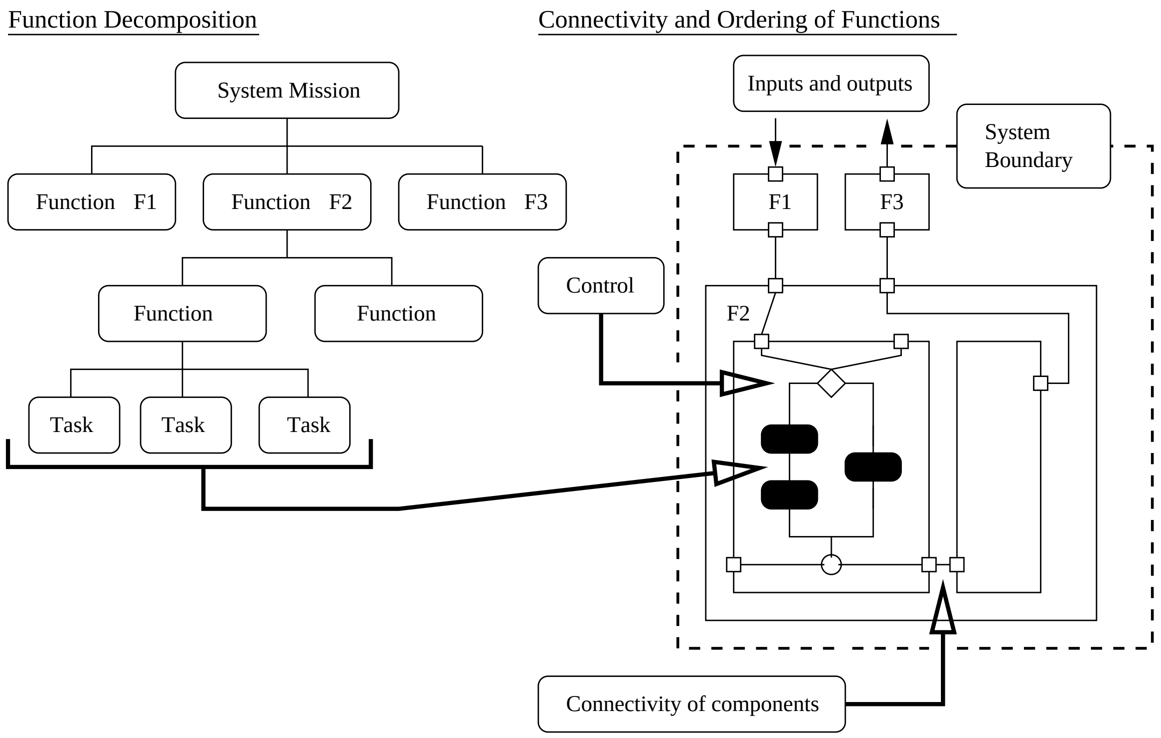
Part 1. Motivation and Approach

Part 1. Motivation and Approach

Motivation and Approach

System Development based on Functional Decomposition

System behavior defined through decomposition and ordering (control) of functions.



Note. The functional decomposition hierarchy says nothing about inputs and outputs.

Motivation and Approach

Functional Approaches to Development

In functional approaches to system development, emphasis is placed on:

1. The transformation of inputs to outputs (with appropriate start and end points), and
2. The systematic decomposition of high-level functions into networks of simpler functionality.

The benefits of functional analysis and design are as follows:

1. The functional approach to development coincides with the way developers naturally look at systems,
2. Top-down functional designs can be tailored to the specific needs of an application.

Examples

Programming in Matlab, C, FORTRAN, ...

Motivation and Approach

Limitations in the Functional Approaches to Development

1. In top-down design, the system is characterized by a single function. This is a questionable concept?
2. Top-down design is based on a functional mind-set. The underlying data types (or data structures) are often ignored.
3. Top-down design by itself does not encourage reusability. Instead, notions of reusabiilty are handled through bottom-up synthesis of previously developed components/concepts.

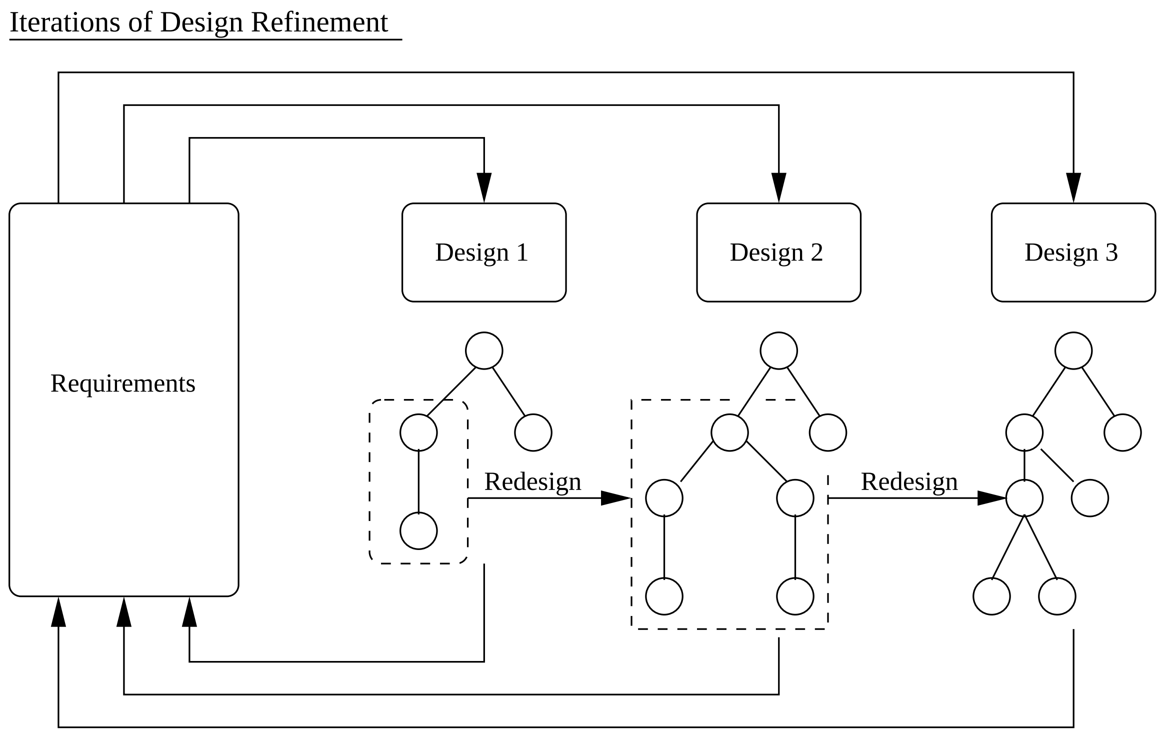
Systems designed along the lines of a top-down functional approach may ...

... not be readily amenable to upgrade and maintenance.

Motivation and Approach

Example 1. Incremental Refinement of a Design

Dealing with change (e.g., change in permissible budget; availability of new and better technologies) is a primary source of difficulty in system development.



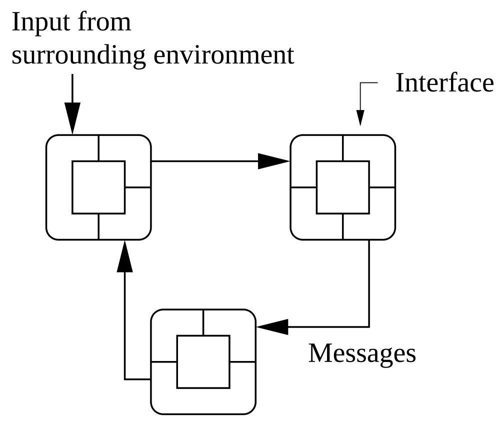
Part 2. Working with Objects

Object-Based Development

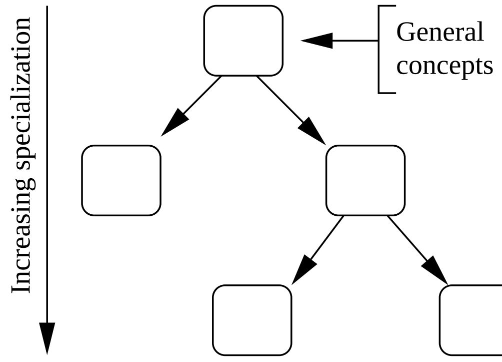
Ideas in Object-Based Development

1. Simplify the way we view the real world,
2. Provide engineers with mechanisms for systematic assembly of complex systems.
3. Claim to provide mechanisms for handling complex problems that are subject to change.

Organizational and Efficiency Mechanisms



Network of Communicating Objects



Problem Domain Concepts organized into a Class Hierarchy.

Object-Based Development

Preliminary Observations for Object-Based Development

- The underlying assumption in object-based development is that ...

  ... real-world systems can be models as networks and hierarchies of objects.

- Object-based systems achieve their purpose with modules having having ...
  - Well defined functionality,
  - Well defined interfaces for connectivity to other modules and the surrounding environment, and message passing.

Design Tasks

- Identify objects and their attributes and functions,
- Establish relationships among the objects,
- Establish the interfaces for each object,
- Implement and test the individual objects,
- Assemble and test the system.

Object-Modeling Techniques

Part 4. Object-Modeling Techniques

Features in Object Modeling

Basic Assumptions

- Everything is an object.
- New kinds of objects can be created by making a package containing other existing objects.
- Objects have relationships for other types of objects.
- Objects have type.

  All objects of the same type can receive and send the same kinds of messages.

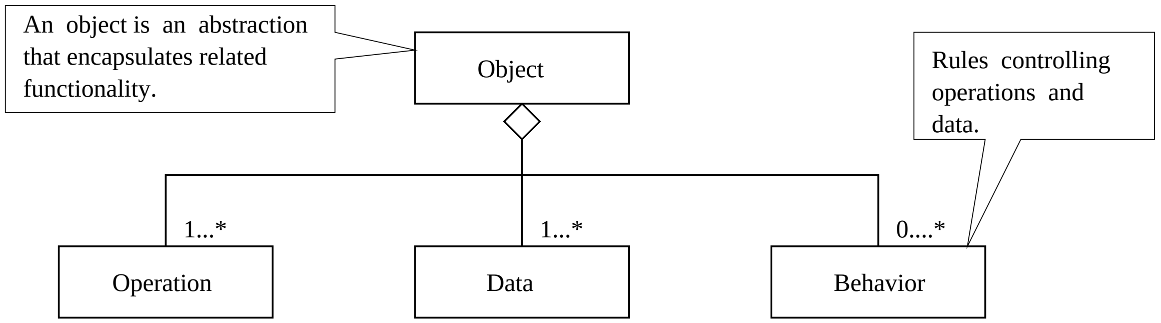
- Objects can have executable behavior.

  They can be design to respond to occurrences and events.

Systems will be created through a composition (assembly) of objects.

Features in Object Modeling

Features of an Object


An object is an abstraction that encapsulates related functionality.
Object
Rules controlling operations and data.
1...*
1...*
0....*
Operation
Data
Behavior

A few key points:

- Objects may be used to represent physical entities, groups of physical entities, or even conceptual entities.
- Object data corresponds to data values held by each object. Common attributes include things like the size, color, cost of the object.
- An object operation is a function or transformation performed on or by a class.
- Rules that specify how the other features of the object are related, or under what conditions the object is viable.

Remarks on Object-Oriented Software

In pure approaches to object-oriented programming:

- **A program is a bunch of objects telling each other what to do.**

  Objects communicate by sending each other messages.

- **Each object has its own memory made up of other objects.**

  New kinds of objects can be packaged from other existing objects.

- **Each object has a type.**

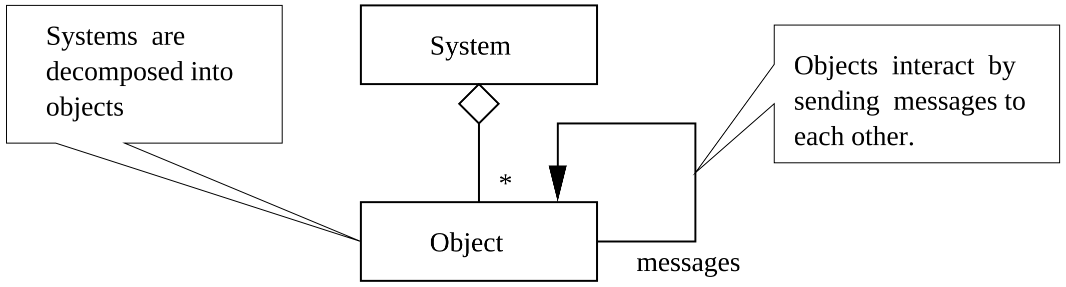
  Each object is an instance of a class. And “class” is synonomous with “type.”

- **All objects of a particular type can receive the same messages.**

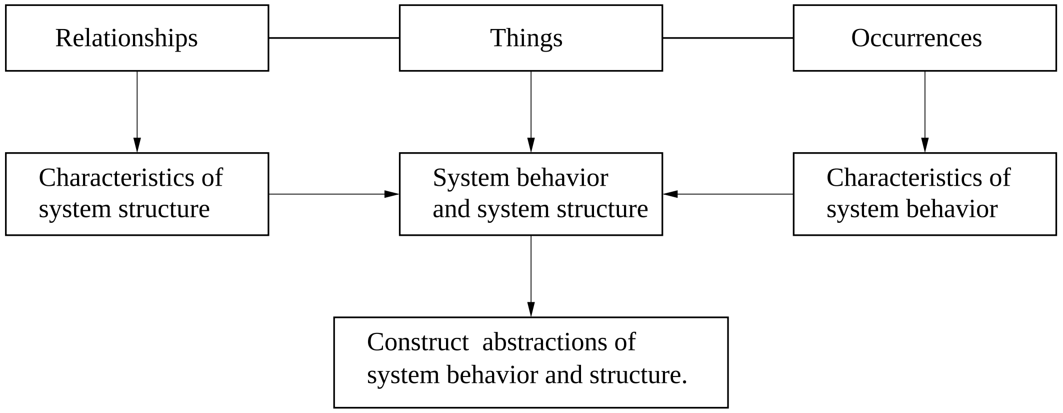
  Because new classes can be created through extensions of existing classes (e.g., an object of type Circle might be created through extension of a Shape class), objects of type Circle and Shape may both be able to receive the same sets of messages.

Synthesis of Object-Based Systems

Systems are created through the Composition of Objects


Systems are decomposed into objects
System
*
Object
messages
Objects interact by sending messages to each other.

Synthesis of System Behavior and Structure


Relationships
Things
Occurrences
Characteristics of system structure
System behavior and system structure
Characteristics of system behavior
Construct abstractions of system behavior and structure.

Synthesis of Object-Based Systems

Things/Objects

- The elements and behavior of engineering systems are the things (or objects) that we care about.

Relationships

- A relationship is a naturally occurring association among specific things.
- Because things (or objects) are built from things, we need a way to express (e.g., counting and enumerating) how things relate to each other and how things within the system are connected to the surrounding environment.

Occurrences/Events

- The important characteristics system behavior required identification of system usage, events, and time sequencing of events.
- An event is an occurrence at a specific time and place.

Object-Oriented Software

External, Temporal and State Events

Systems engineers need to plan for and react to the following types of events:

Event Type	Description
External Events	These are events that occur outside the system boundary. For most systems, external events trigger a set of actions the system must response to.
Temporal Events	These are events that occur as the result of reaching some point in time.
State Events	A state event occurs when something happens inside the system that triggers the need for processing.

Object-Oriented Software

Object-Oriented Development Process

Object-oriented development procedures observe that in real life:

- Collections of objects share similar traits. They may store the same data and have the same structure and behavior.
- Then, collections of objects will form relationships with other collections of objects.

Instead of working in terms of objects alone, it makes sense to create models that capture the common attributes, properties and behaviors shared by collection of objects.

Definition of a Class

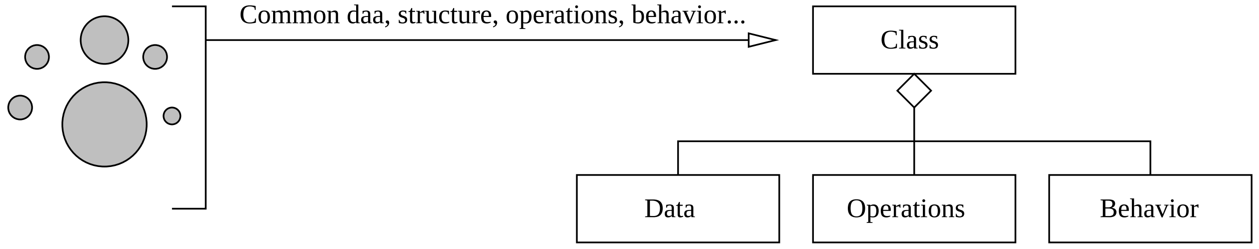
A class is ...

... a specification (or blueprint) of an object's behavior and structure.

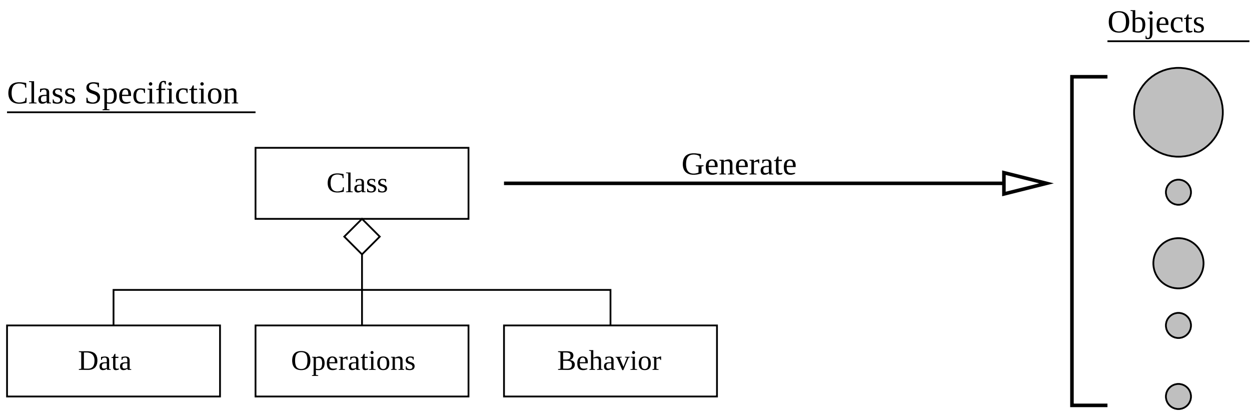
Each object is an instance of a class.

Objects and Classes

Pathway from Collections of Objects to Classes

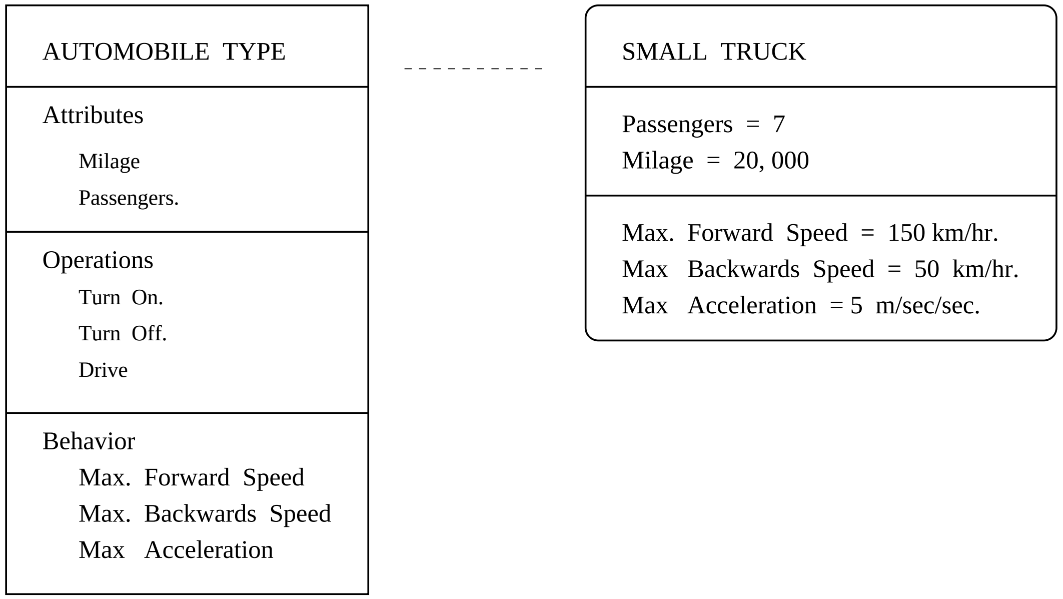


Generation of Objects from a Class Specification



Objects and Classes

Example. Automobile Class and Automobile Object



Remark. This figure is drawn in OMT (an acronym for object modeling technique), a visual language developed in 1991 by Rumbaugh et al. to support object-oriented systems and object-oriented programming.

Objects and Classes

Points to note.

The class is partitioned into four areas:

1. The top area contains the name of the class.
2. The attributes of the class. No values are assigned to the attributes.
3. Next come the operations and behavior of the class.

The object is partitioned into three areas:

1. The name of the object.
2. The object data. Notice that the data values have now been instantiated.
3. Rules and methods that define the object behavior.

At the object level, the values of the attributes are what distinguishes one instance of the class (i.e., an object) from another instance. Also notice that the object operations are kept at the class level.

Objects and Classes

Example 1. A Simple Class in Java

```
public class Point {
   int x, y;

   public Point ( int x, int y ) {
      this.x = x; this.y = y;
   }
}
```

Creating an Object

```
Point  first = new Point ( 1, 2 );
Point second = new Point ( 2, 5 );
```

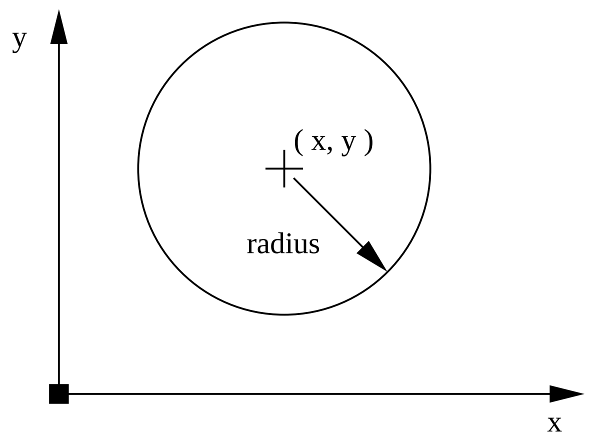
Accessing and Printing the attributes on an Object

```
System.out.printf(" first point (x,y) = (%2d, %2d)\n",  first.x,  first.y );
System.out.printf("second point (x,y) = (%2d, %2d)\n", second.x, second.y );
```

Objects and Classes

Example 2. Working with Circles

A circle can be described by the `x` and `y` position of its center and by its `radius`.



There are numerous things we can do with circles ...

- Compute their circumference or perimeter,
- Compute their area,
- Check if a point is inside a circle.

Objects and Classes

Example 2. Working with Circles

```
// =================================================================
// Circle.java: This class defines circles.
// =================================================================

import java.lang.Math.*;

public class Circle {
    double dX, dY, dRadius;

    // Constructor methods ....

    public Circle() {}

    public Circle( double dX, double dY, double dRadius ) {
        this.dX = dX;
        this.dY = dY;
        this.dRadius = dRadius;
    }
    .....
}
```

Objects and Classes

Example 2. Working with Circles

```
    ....

    public double Area() {
       return Math.PI*dRadius*dRadius;
    }

    public String toString() {
       return "(x,y) = (" + dX + "," + dY + "): Radius = " + dRadius;
    }

    public static void main( String [] args ) {
       Circle cA = new Circle( 1.0, 2.0, 3.0 );
       System.out.println("Circle cA : " + cA.toString() );
       System.out.println("Circle cA : Area  = " + cA.Area() );
    }
}
```

Objects and Classes

Example 2. Script of Program Input/Output

```
Script started on Thu Nov 03 07:43:33 2005
prompt >>
prompt >> java Circle
Exercise methods in class Circle
================================
Circle cA : (x,y) = (1.0,2.0): Radius = 3.0
Circle cA : Area  = 28.274333882308138
prompt >>
prompt >> exit
script done on Thu Nov 03 07:43:49 2005
```

Object Data and Methods

Accessing Object Data

Now that we have created an object, we can use its data fields. The ...

> **... dot operator (.) is used to access the different public variables of an object.**

For example

```
    Circle smallCircle = new Circle();

    /* Initialize the circle to have center (2,2) and radius 1.0 */

    smallCircle.dX = 2.0;
    smallCircle.dY = 2.0;
    smallCircle.dR = 1.0;
```

Object Data and Methods

Accessing Object Methods

To access the methods of an object, we use the ...

... same syntax as accessing the data of the object, i.e., the dot operator (.).

Example 1

```
Circle smallCircle = new Circle();
smallCircle.dR = 2.5;
double dArea   = smallCircle.area();
```

Notice that we did not write

```
dArea = area( smallCircle );
```

Example 2

Let a, b, c, and d be complex numbers. To compute a*b + c*d we write

```
a = new Complex(1,1); .. etc ..

Complex sum = a.Mult(b).Add( c.Mult(d) );
```

Encapsulation and Data Hiding

Definition of Aggregation

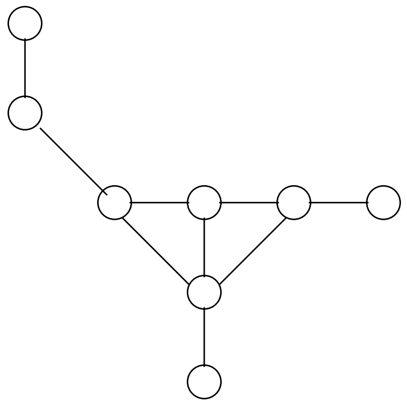
- Aggregation is the grouping of components into a package.
- Aggregation does not imply that the components are hidden or inaccessible.
- Instead, aggregation merely implies that the components are part of a whole.

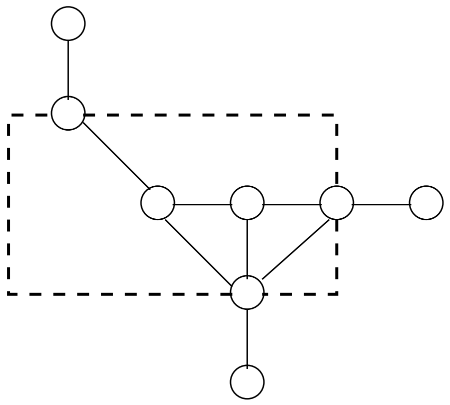
Definition of Encapsulation

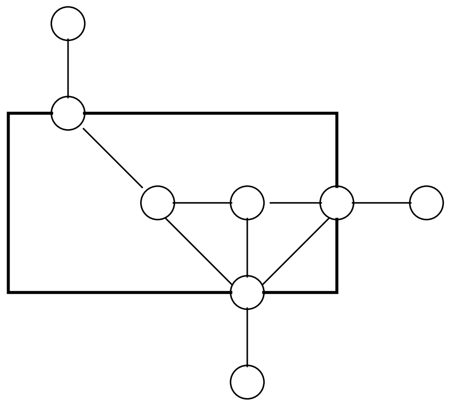
- Encapsulation is a much stronger form of organization.
- Encapsulation forces users of a system to deal with it as an abstraction (e.g., a black box) with well-defined interfaces that define what the entity is, what it does, and how it should be used.
- The only way to access an object's state is to send it a message that causes one of the object's internal methods to execute.

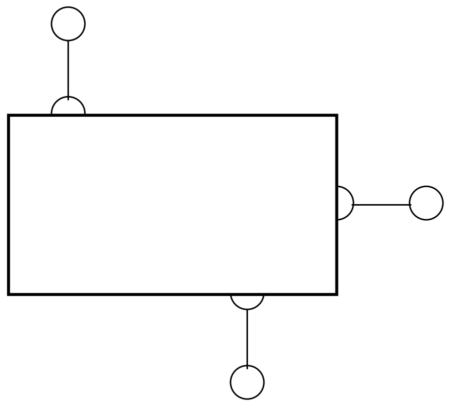
Encapsulation and Data Hiding

Schematic for Unstructured Components, Aggregation and Encapsulation


Unstructured Components


Aggregation


Designer’s view of Aggregation


Encapsulation −− User’s view of Abstraction

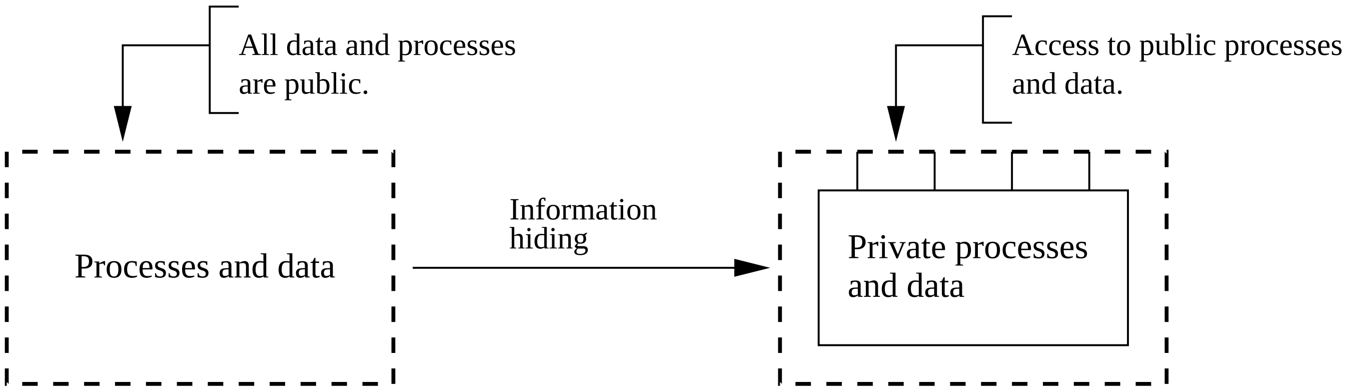
Encapsulation and Data Hiding

Principle of Information Hiding

The principle of information hiding states that ...

> **... information which is likely to change (e.g., over the lifetime of a software/systems package) should be hidden inside a module.**

Application. Process for Implementation of Information Hiding.

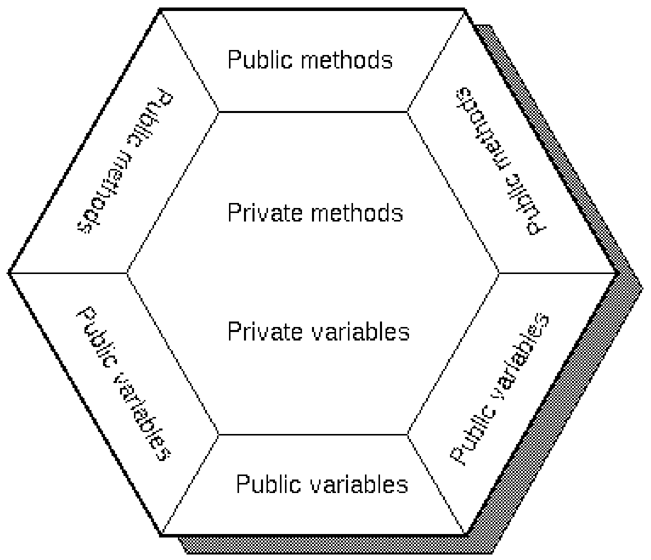

All data and processes are public.
Access to public processes and data.
Processes and data
Information hiding
Private processes and data

Note. The information to be hidden could be a companies intellectual property.

Encapsulation and Data Hiding

Graphical Representation of a Class

Graphical representation of a Class



The object wrapping ...

... protects the object code from unintended access by other code.

Encapsulation and Data Hiding

In object-oriented terminology, and particularly in Java,

1. The wrapper object is usually called a **class**, the functions inside the class are called **private methods**,
2. The data inside the class are **private variables**.
3. **Public methods** are the interface functions for the outside world to access your private methods.

Information Hiding

Implementation of Information Hiding

The keyword **private** in:

```
public class Point {
   private int x, y;
   .... .....
}
```

restricts to scope of x and y to lie inside the boundary of Point objects.

Access to a point’s coordinates is controlled through the public methods:

```
public int  getX() {
   return x;
}
public void setX(int x) {
   this.x = x;
}
```

Relationships Among Classes

Definition

Classes and objects by themselves are ...

> **... not enough to characterize requirements or design a system.**

We also need a way to express relationships among classes.

> **As association is a discrete/logical relationship between classes.**

In order for the association to work, ...

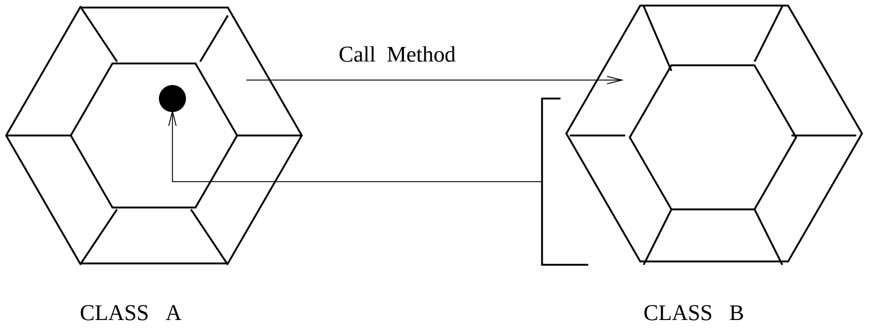
> **... each of the participating classes must be aware of the association's existence.**

Associations are the glue that tie the elements of a system together.

Relationships Among Classes

Object-oriented software packages are assembled from collections of classes and class-hierarchies that are **related in three fundamental ways.**

1. **Use:** Class A uses Class B (method call).



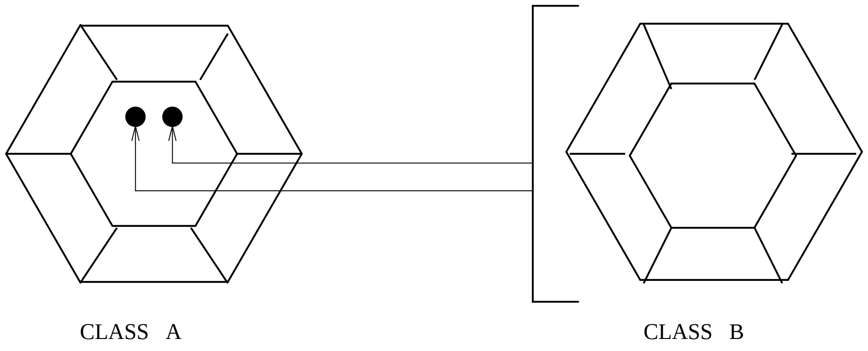
Class A uses Class B if a method in A calls a method in an object of type B, or alternatively, a method of A creates, receives, or returns objects of type B.

Example

```
double dAngle = Math.sin ( Math.PI / 3.0 );
```

Relationships Among Classes

2. **Containment (Has a):** Class A contains a reference to Class B.



Clearly, containment is a special case of use (i.e., see Item 1.).

Example

```
public class LineSegment {
   private Point start, end;

   .......
}
```

Relationships Among Classes

3. **Inheritance (Is a):** In everyday life, we think of inheritance as something that is received from a predecessor or past generation. The physical features we inherit from our ancestors are perhaps the best example of inheritance.

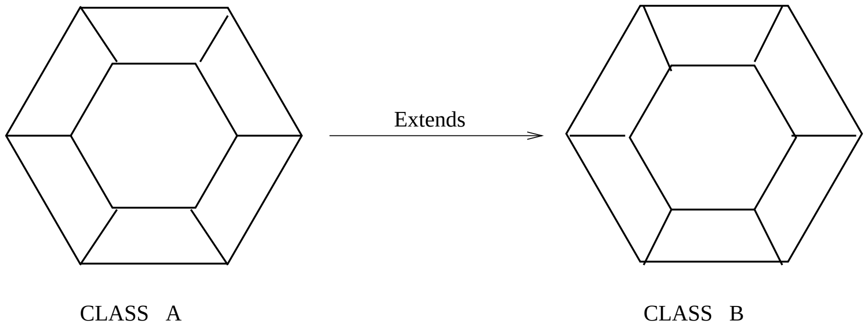


Figure 1: Class B inherits the data and methods (extends) from Class A.

Examples of Java Code

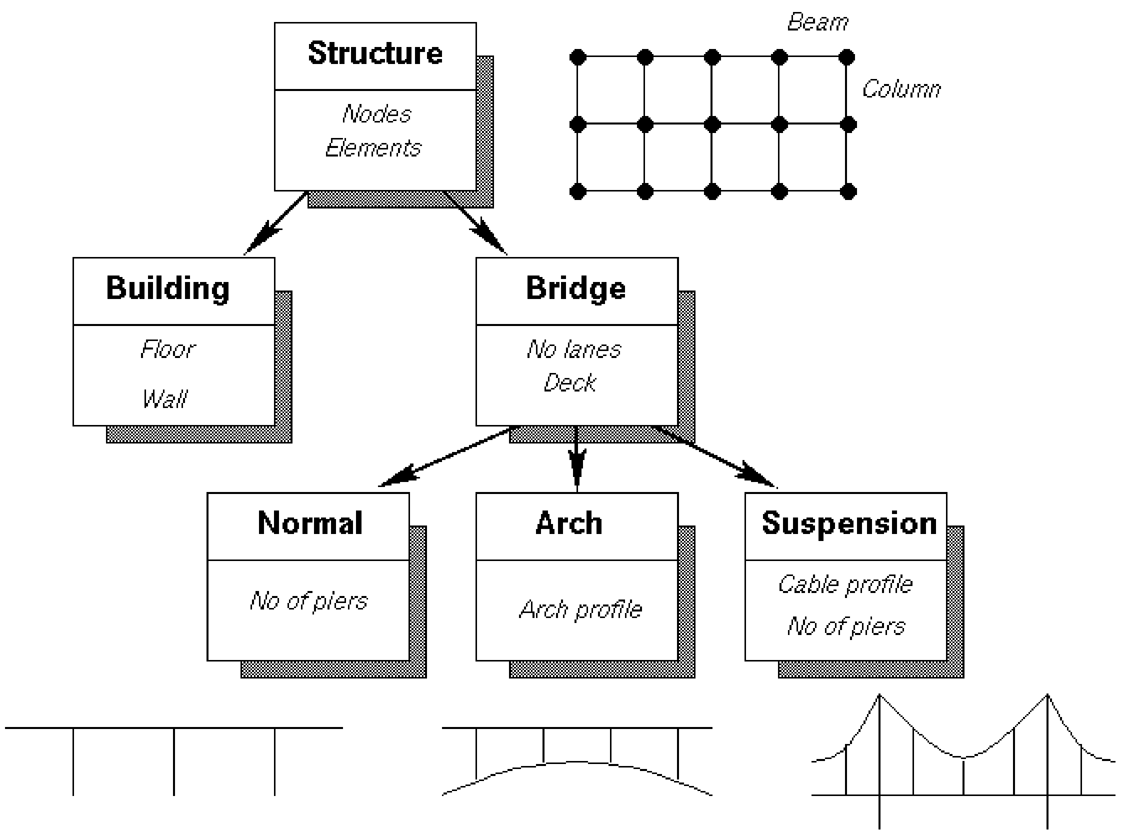
```
public class ColoredCircle extends Circle { .... }

public class GraphicalView extends JFrame { .... }
```

Relationships Among Classes

Inheritance in Modeling of Building and Bridge Structures

Civil engineering structures are often modeled as an assembly of nodes and elements.

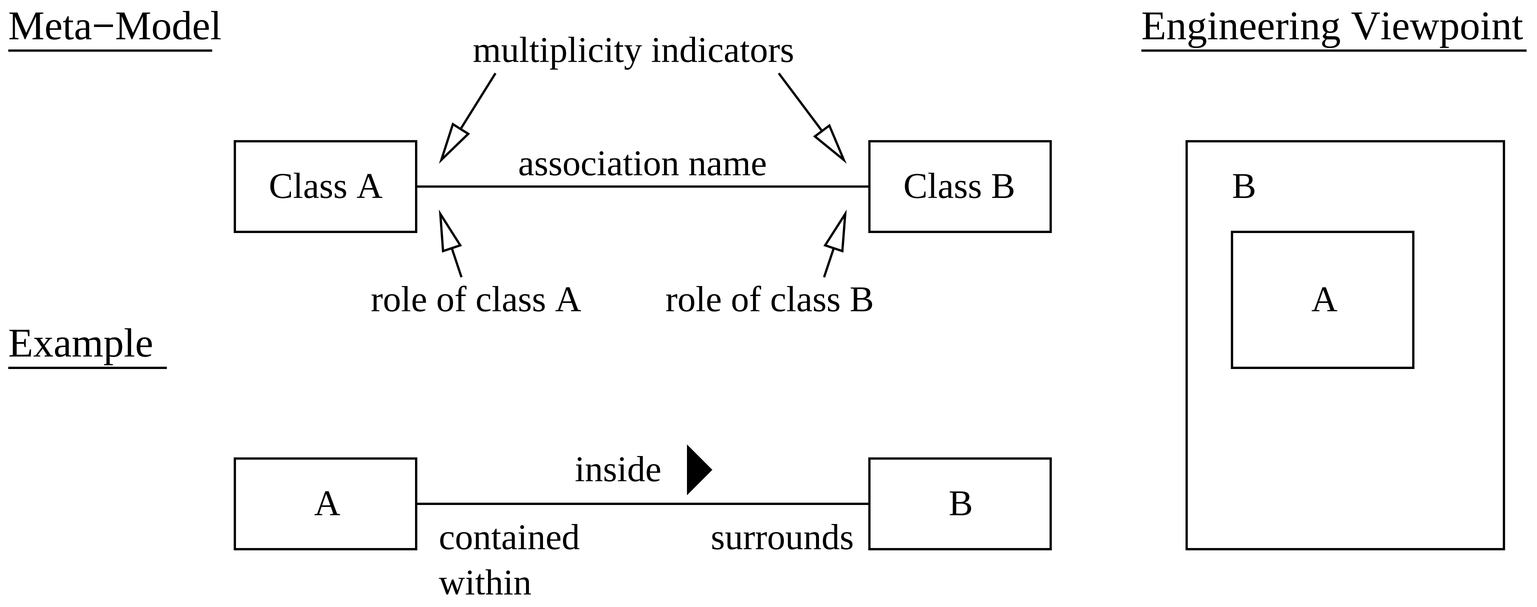
Inheritance in Modeling a Structure



Binary Association Relationships

Example. Binary Associations

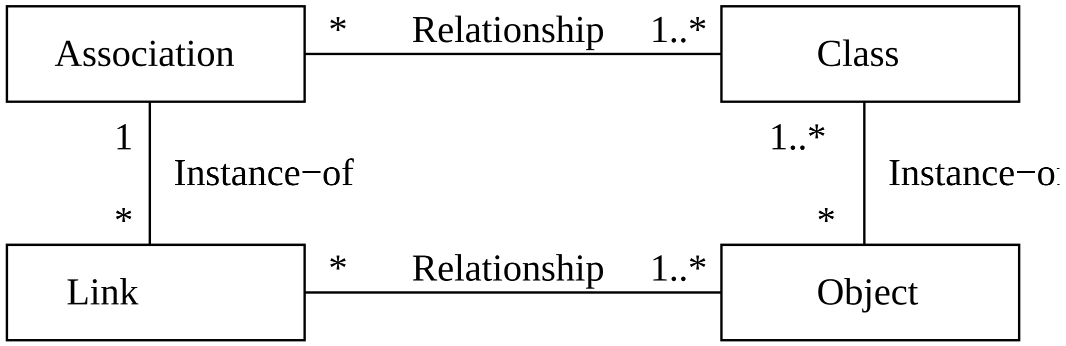
Binary associations express static a bidirectional relationships between two classes.



Association Relationships

Meta-Model for Links and Association Relationships

Links and associations establish relationships among entities within the problem world or the solution world.



Points to note:

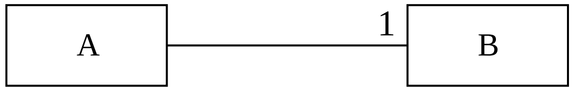
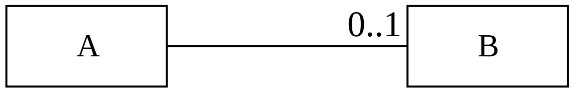
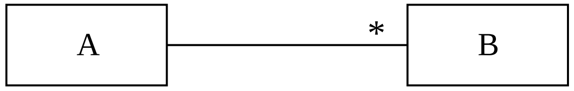
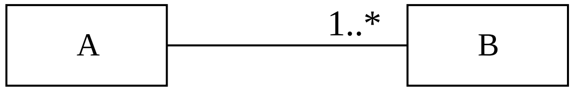
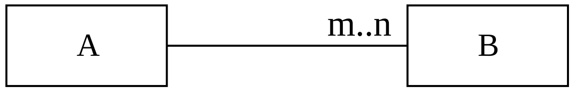
- Associations are descriptions of links with a common implementation.
- Links are instances of an association relationship.

Put another way, an association specifies how an object type is specified in terms of other object types (see Graham, pg's 12-30; pg. 251).

Association Relationships

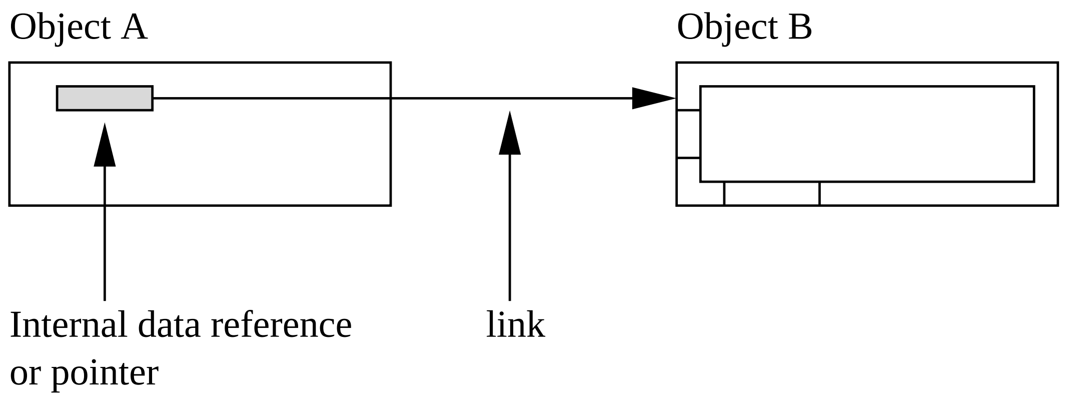
Multiplicity Constraints

Indicate the number of objects participating in a particular instance of an association.

Relationship	Multiplicity
	Exactly one to one
	Optional (zero or one)
	Many (zero or more)
	Many (one or more)
	Numerically specified

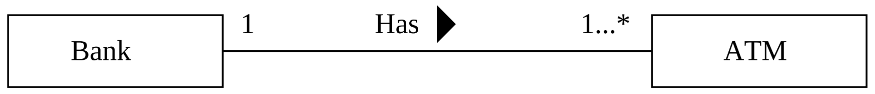
Association Relationships

Example 1. Symbolic Representation for Object A linking to Object B



Example 2. One-to-Many multiplicity between a bank and a suite of ATMs

Consider the relationship between a bank and an ATM.

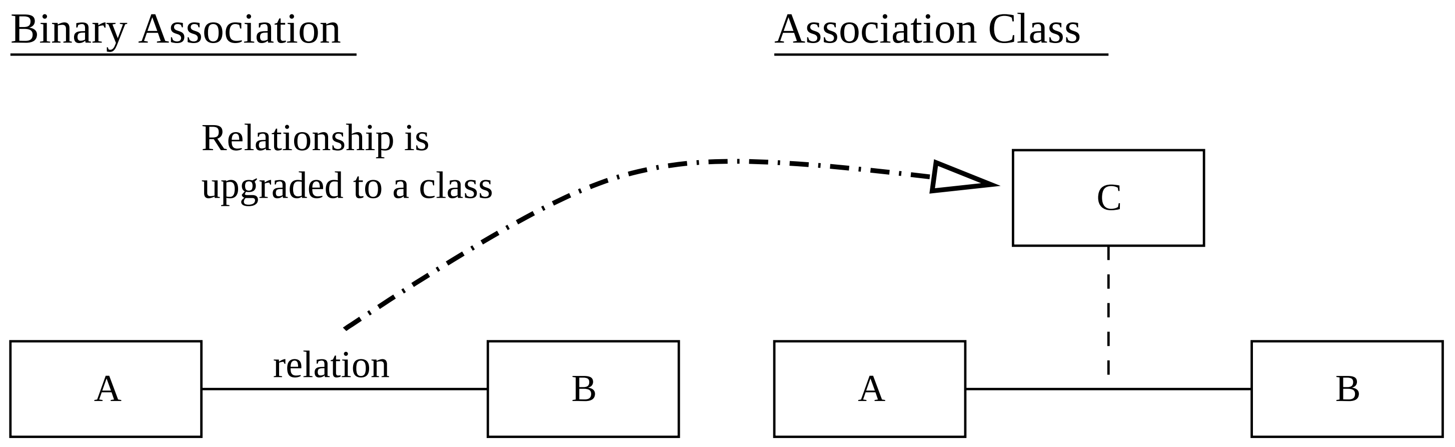


The diagrams states:

- A bank has one or more ATMs.
- Each ATM is associated with one (and only one) bank.

Association Relationships in UML

From Binary Relations to Association Classes



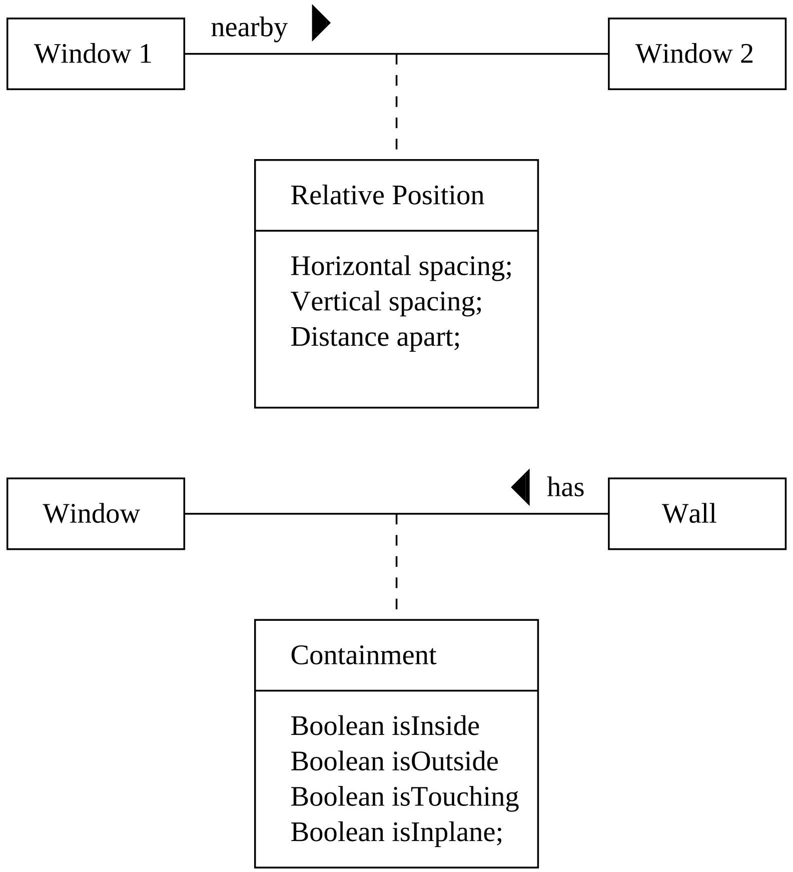
Association Class Relationships

Definition

Association classes are used when:

- The association itself has attributes or operations that need to be represented in the class model.
- It makes sense for the "one association occurrence, one association class instance" constraint to exist.

Two examples:



Inheritance Mechanisms

Inheritance structures allow you to capture common characteristics in one model artifact and permit other artifacts to inherit and possibly specialize them.

- This approach to development forces us to identify and separate the common elements of a system from those aspects that are different/distinct.
- The commonalities are captured in a super-class and inherited and specialized by the sub-classes.

What's really cool is that ...

... inherited features may be overridden with extra features designed to deal with exceptions.

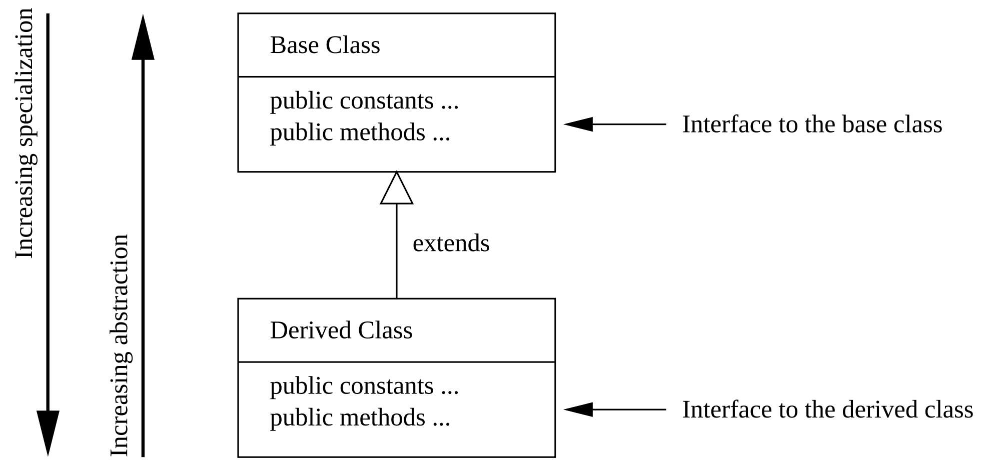
This setup implies that ...

... class hierarchies must be explicitly designed for customization through extension.

Inheritance Mechanisms

Example 1. Base and Derived Classes

Goal: Avoid duplication and redundancy of data in a problem specification.



A class in the upper hierarchy is called a superclass (or base, parent class).

A class in the lower hierarchy is called a subclass (or derived, child, extended class).

The classes in the lower hierarchy ...

...inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher hierarchies.

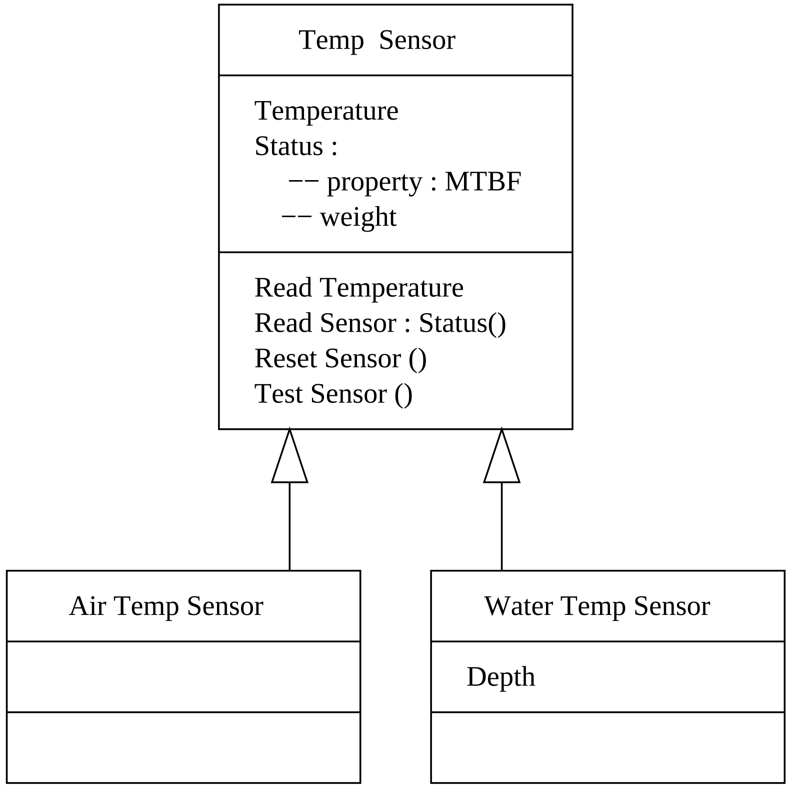
Inheritance Mechanisms

Example 2. Temperature Thermometer

- Consider a class hierarchy for attributes and functions in a family of temperature sensors.
- The super-class represents a generic temperature sensor.
- Super-class attributes: measured temperature, sensor weight, mean-time-to-failure (MTTF).
- Methods are provided to test the sensor.

Water Temperature Thermometer

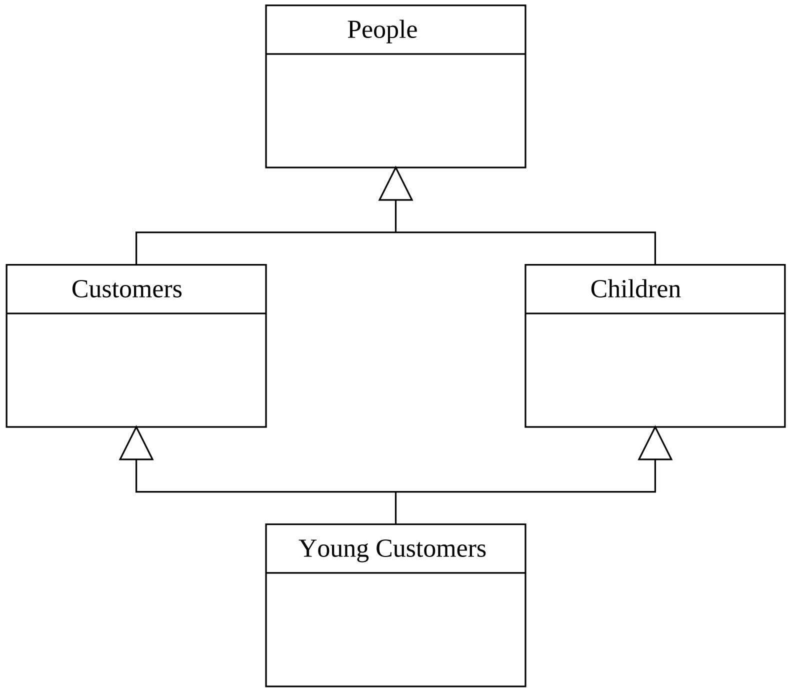
- A water temperature thermomenter is a generic temperature sensor + a field to store the depth at which the temperature was recorded.



Inheritance Mechanisms

Multiple Inheritance Structures

- In a multiple inheritance structure, a class can inherit properties from multiple parents.
- The downside is that properties and/or operations may be partially or fully contradictory.



Example

- People is a generalization of Children and Customers.
- Young customers inherits properties from Customers and Children.

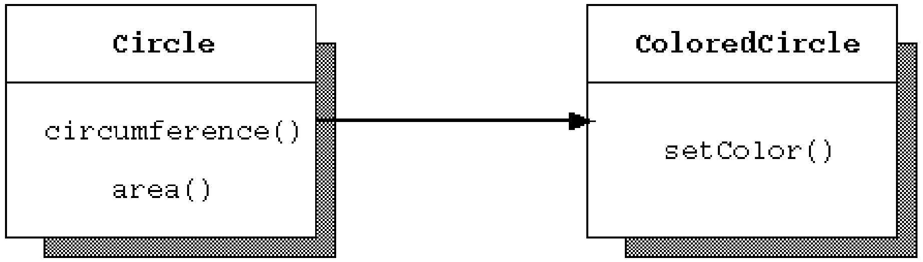
Inheritance Mechanisms

Example 3. Extending Circle to create Colored Circle

```
public class ColoredCircle extends Circle {
      private Color color;  // The color of the circle

      // Constructor method for this class.

      public ColoredCircle() {
         super(); // Call the superclass constructor method
         this.color = Color.black;
      }

      // Set the color for the current circle.

      public void setColor(Color c) {
         color=c;
      }
}
```

Inheritance Mechanisms

Example 3. Extending Circle to create Colored Circle



Two public methods are defined for this class:

- `setColor.` This method takes a color as its argument and assigns this value to the color of the circle.
- `ColoredCircle.` This method has the same name as the class itself; it is a constructor method.

The method call `super()` invokes the constructor method of the superclass [i.e., the method `Circle()`].

Aggregation and Composition

Definition of Aggregation

- Aggregation relationships indicate how classes/things are included in (or used) to build other classes/things.
- Aggregation is also known as ...

  ... a “has a” relationship

  because the containing object has a member object and the member object can survive or exist without the enclosing or containing class or can have a meaning after the lifetime of the enclosing object.

Definition of Composition

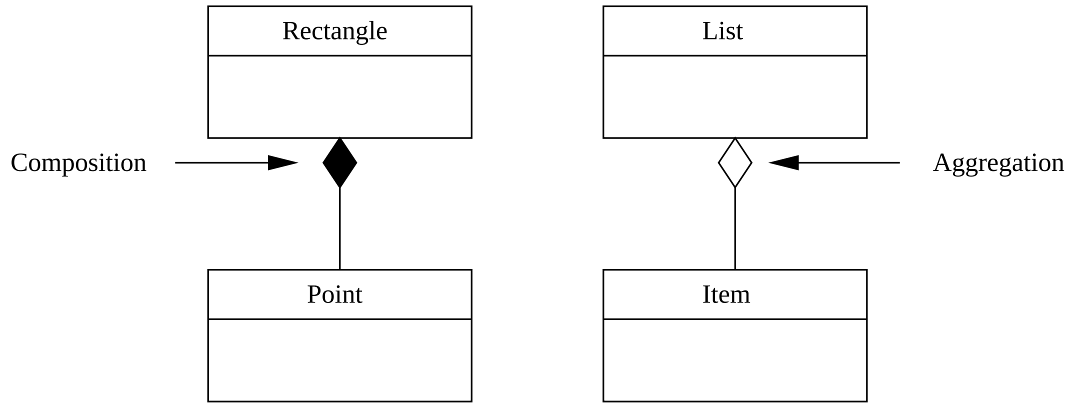
- Composition is also known as ...

  ... a “is a part of” or “is a” relationship

  because the member object is a part of the containing class and the member object cannot survive or exist outside the enclosing or containing class or doesnt have a meaning after the lifetime of the enclosing object.

Aggregation and Composition

Notation for Aggregation and Composition



Test: Aggregation or Composition?

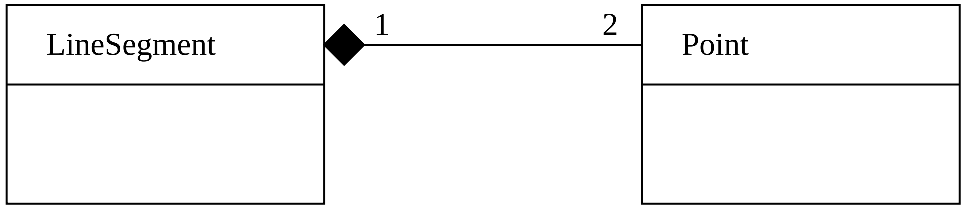
- A good way of identifying a composition relationship (Binder, 2001) is to ask the question:

  ... if the part moves, can one deduce that the whole moves with it in normal circumstances?

- Example. A car is composition of wheels and an engine. If you drive the car to work, hopefully the wheels go too!

Aggregation and Composition

Example 1. A LineSegment is composed from two instances of a Point class.



```
public class Point {
    private int x, y;
    public Point(int x, int y) {
       this.x = x; this.y = y;
    }
    public int  getX()      { return x; }
    public void setX(int x) { this.x = x; }
    public int  getY()      { return y; }
    public void setY(int y) { this.y = y; }
    public String toString() {
       return "(" + x + "," + y + ")";
    }
}
```

```
public class LineSegment {
   Point begin, end;
   public LineSegment (int x1, int y1,
                       int x2, int y2) {
      begin = new Point(x1, y1);
      end   = new Point(x2, y2);
   }

   public String toString() {
      return "Line segment: from " +
              begin + " to " + end;
   }
}
```

Aggregation and Composition

Implementation of Information Hiding

- The keyword **private** in:

```
public class Point {
   private int x, y;
   .... .....
}
```

  restricts to scope of x and y to lie inside the boundary of Point objects.

- Access to a point’s coordinates is controlled through the public methods:

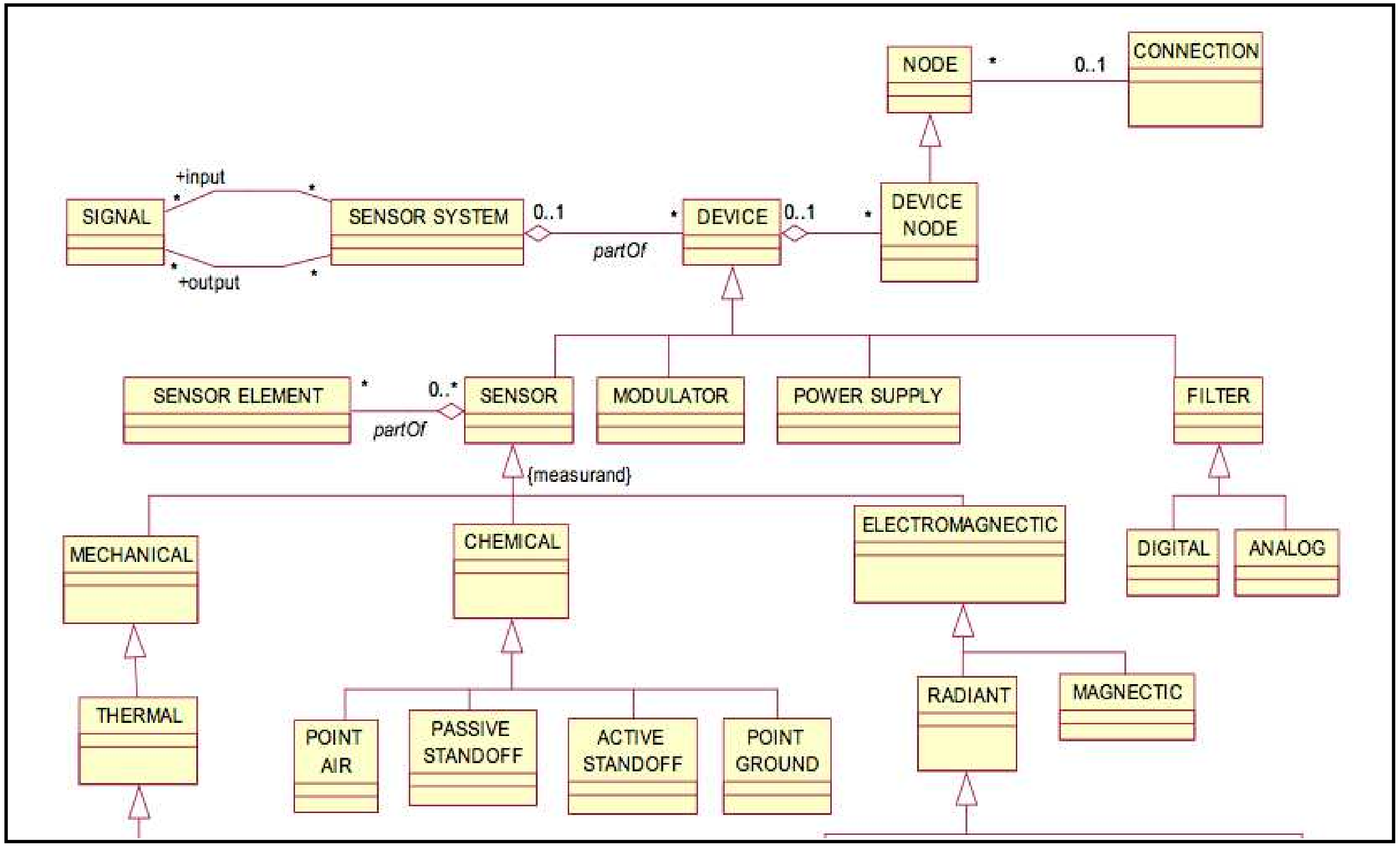
```
public int  getX() {
   return x;
}
public void setX(int x) {
   this.x = x;
}
```

Part 4. Applications

Part 4. Applications

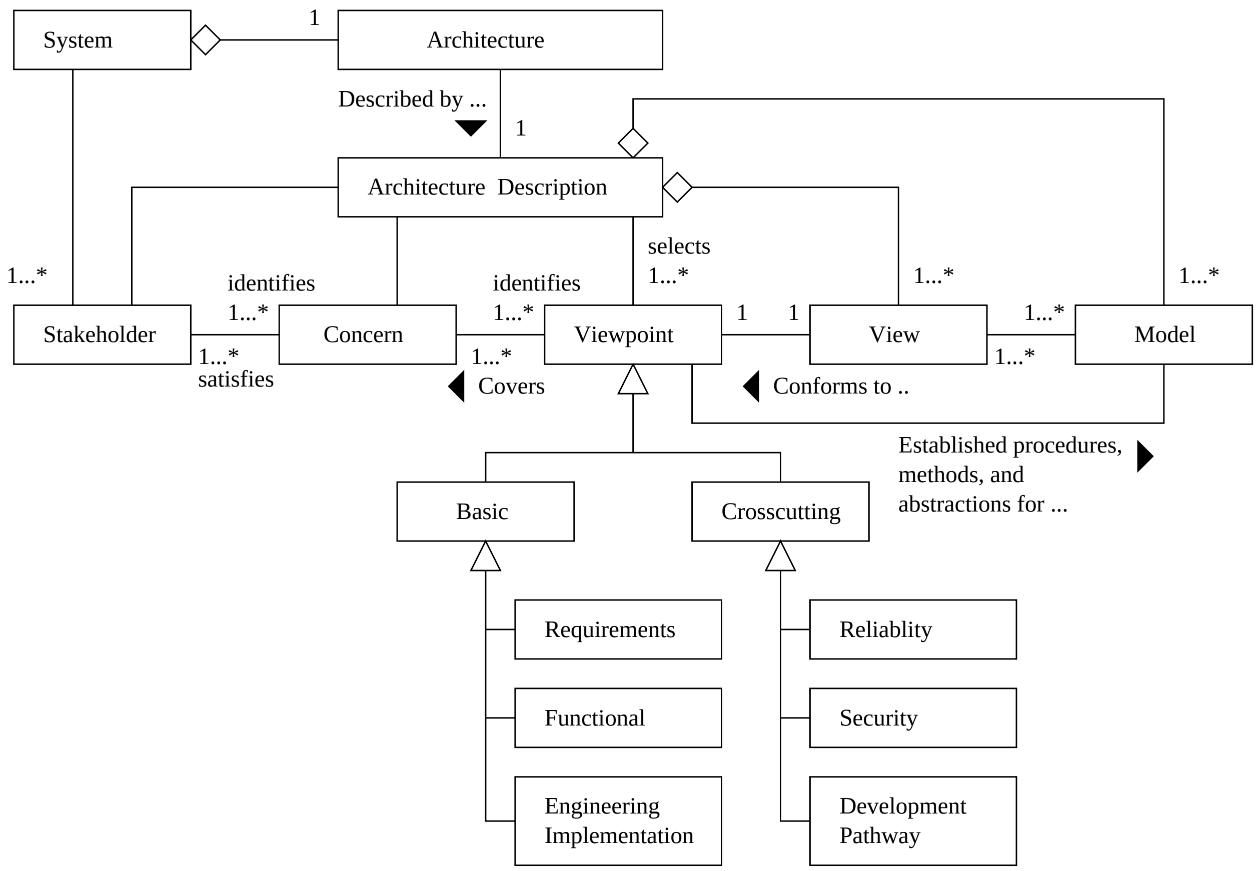
Class Diagram for Sensors

Class Diagram for Sensors



System Development Framework

Example 1. Systems Development Framework for Multiple Stakeholders



Assembled from ideas due to Eeles et al. (2010), Maier (1998), and definitions in the IEEE 1471 Standard.

System Development Framework

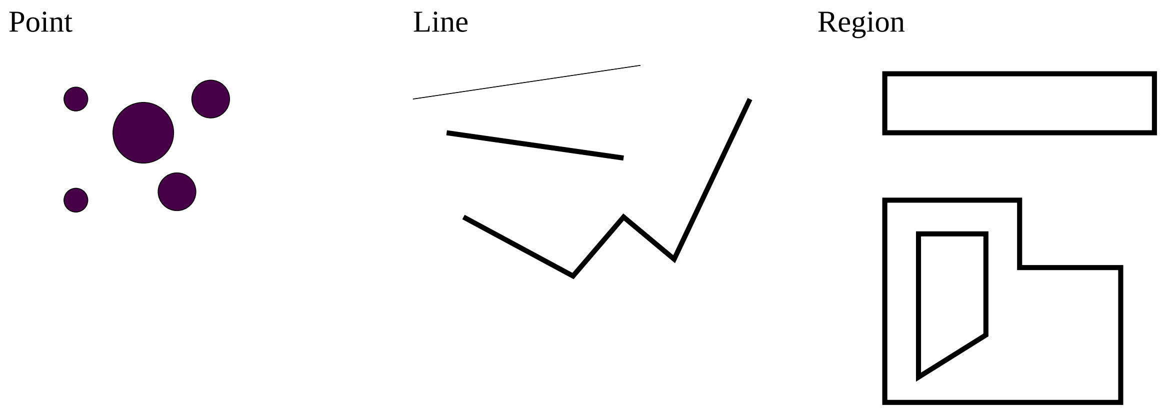
Points to note.

- An architecture is a fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- A system stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.
- Typical design concerns include system functionality, performance, reliability, security, distribution, ease of evolvability, schedule of development, maintenance and cost.
- A view is a representation of a whole system from the perspective of a related set of concerns.
- A viewpoint is a specification of the conventions (i.e., languages and models) for assembling and using a view.
- Viewpoints may be partitioned into basic viewpoints and crosscutting viewpoints.

Class Diagram for GIS Domain

Example 2. Points, Lines and Regions for GIS

Points, lines and regions are fundamental spatial data types.



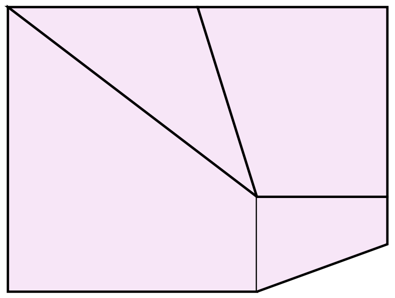
- Points are 0-dimensional entities. Lines are 1-dimensional entities. Regions are 2-dimensional entities.
- Spatial operations: union, intersection, difference.
- We need software that can compute operations on these entities in a consistent manner (e.g., google: Java Topology Suite).

Class Diagram for GIS Domain

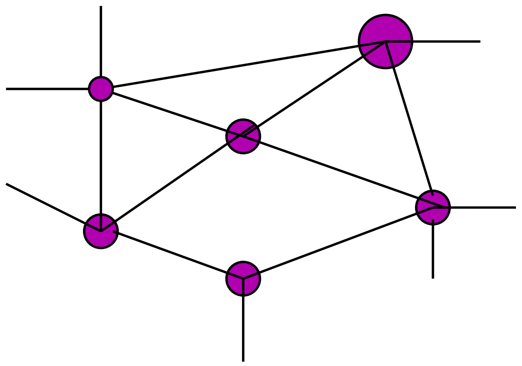
Partitions and Networks

Partitions and networks are two abstractions for modeling collections of spatial objects.

Partitions



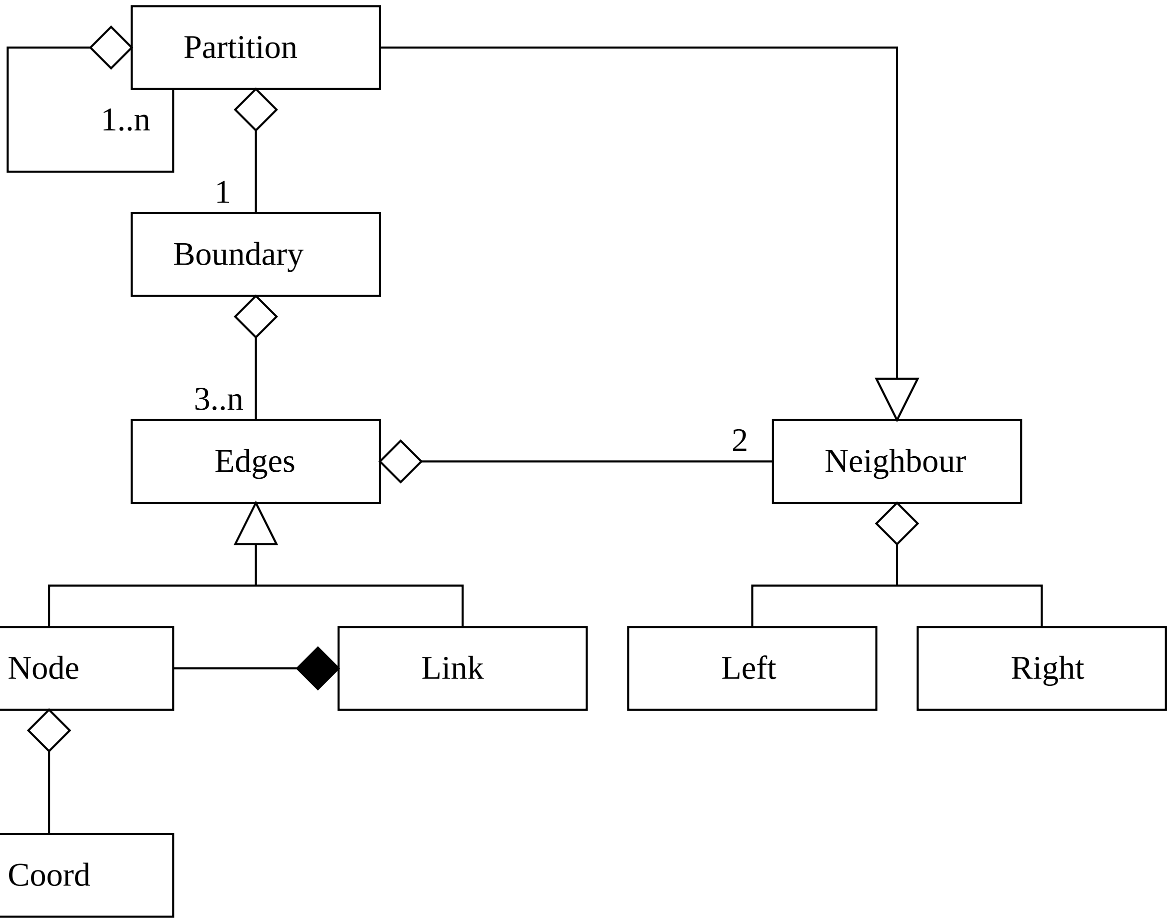
Spatially Distributed Network



- Examples of partitions: rooms in a building, districts in a state, countries in a continent.
- Examples of networks: plumbing and HVAC networks, highways and railway networks, communication and power networks.

Class Diagram for GIS Domain

Conceptual model for partition hierarchies (adapted from Chunithipaisanl S. et al., 2004)



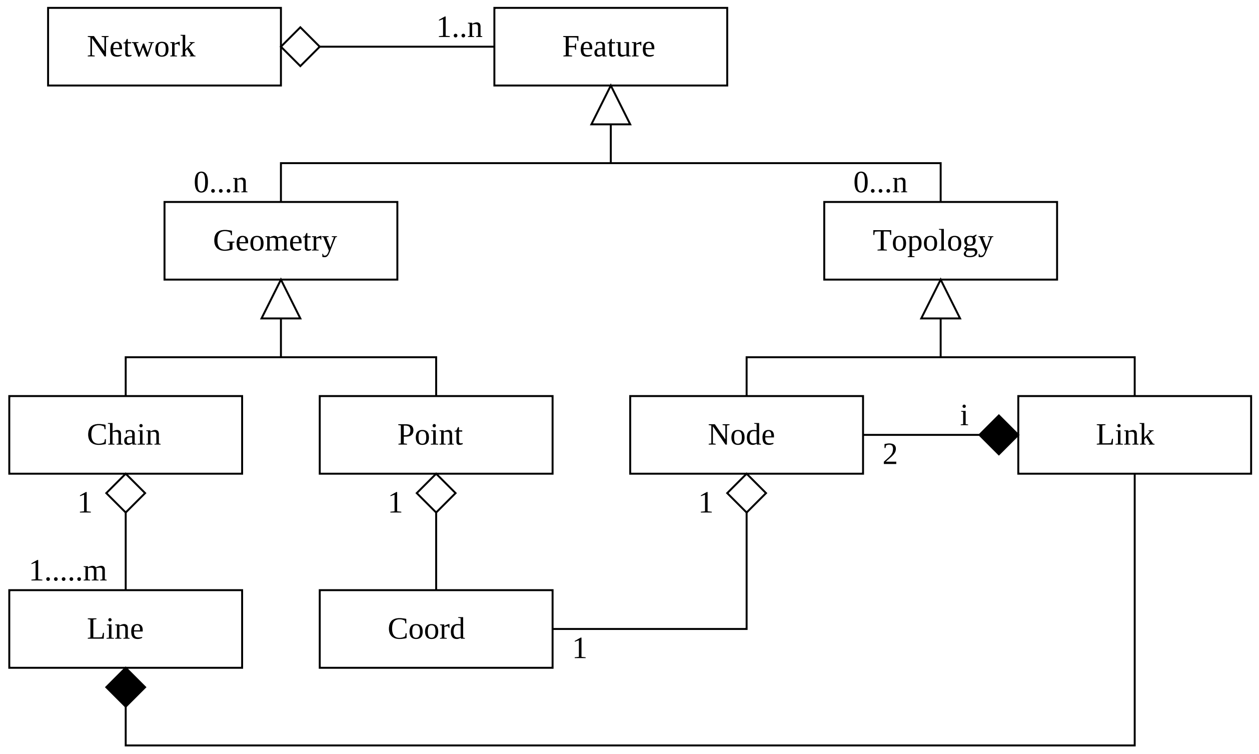
Class Diagram for GIS Domain

The conceptual model for partitions states:

1. A Partition can be decomposed into 1 or more Partitions (sub-Partitions).
2. Each Partition has one boundary (here we ignore the possibility of partitions containing holes).
3. Boundaries are composed of edges (..at least 3 edges).
4. Each Edge segment has a Node and Link.
5. Nodes and Link are paired in a one-to-one correspondence.
6. A Node has a coordinate.
7. Edges also have Neighboring Partitions.
8. Neighboring Partitions can be classified as to whether they are on the Left and Right of the Edge.

Class Diagram for GIS Domain

Conceptual model for networks (Adapted from: Chunithipaisanl S. et al., 2004).



Class Diagram for GIS Domain

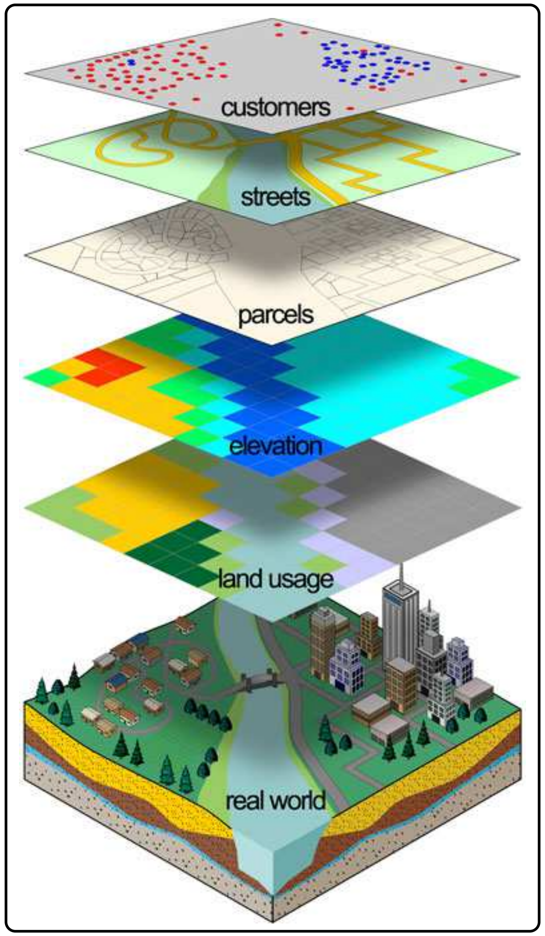
The conceptual model for networks states:

1. A Network is composed of Features.
2. Each Feature has Geometry and Topology.
3. Geometry is a generalization for Chains and Points...
4. A Chain corresponds to one or more Line segments.
5. A Point has a coordinate.
6. Topology is a generalization for Nodes and Links.
7. Nodes also have coordinates.

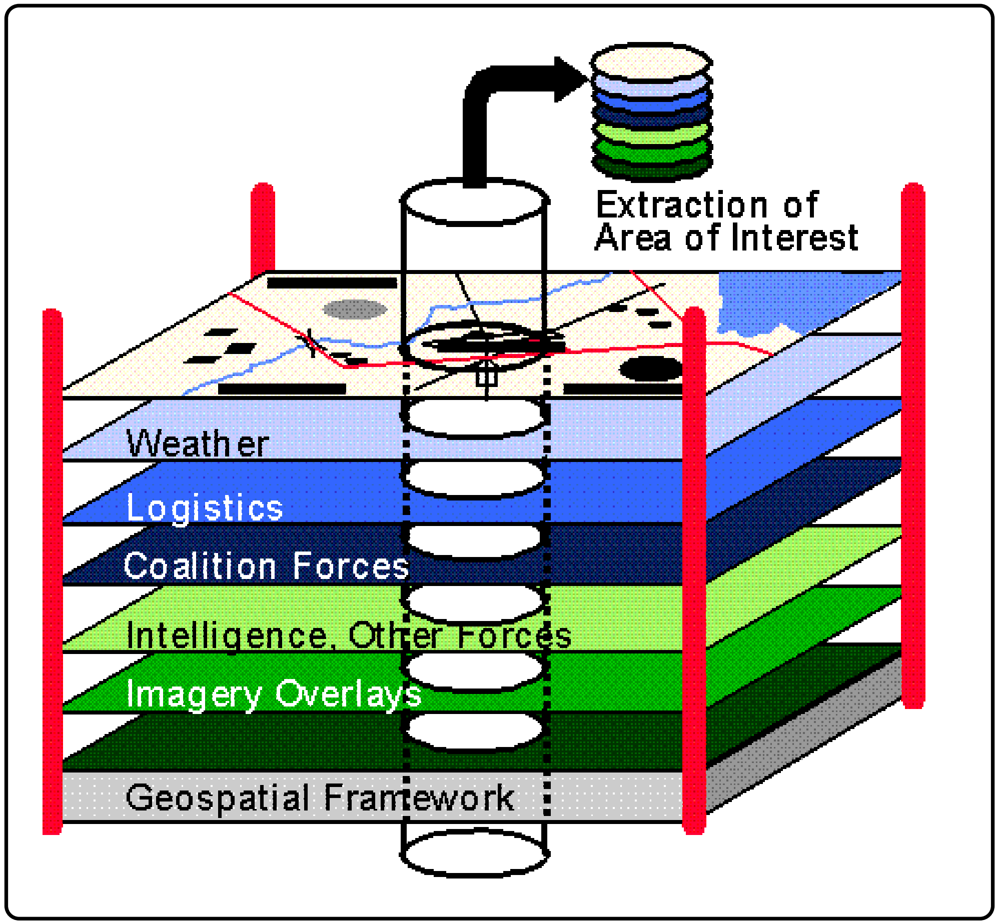
Layers of Spatial Data

Example. Layered organization of multi-dimensional attributes in spatial data.

Geographic Information System



Layers of Data / Information in Military Decision Making



Class Diagram for GIS Domain

Ph.D. Qualifying Exam in Civil Systems (January 2013)