Java Tutorial: Working with Objects and Classes

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Fall Semester 2020

October 10, 2020

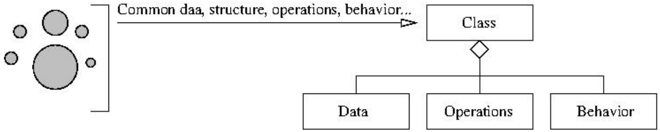
Overview

Part 2

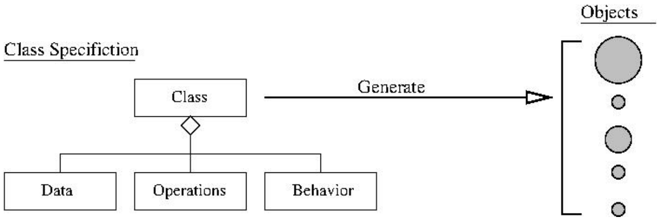
Working with Objects

Working with Objects and Classes

From Collections of Objects to Classes:

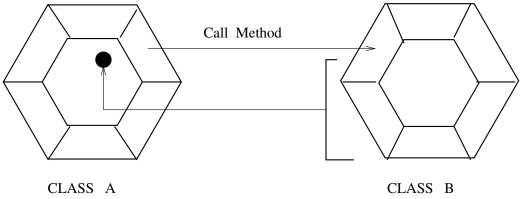


Generation of Objects from Class Specifications:



Relationships Among Classes

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



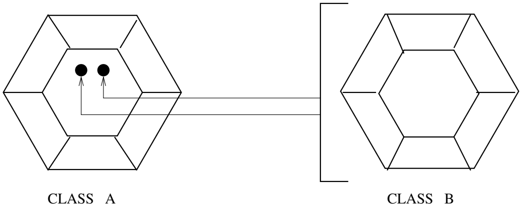
Class `A` uses Class `B` if a method in `A` calls a method in an object of type `B`.

Example

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

Relationships Among Classes

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



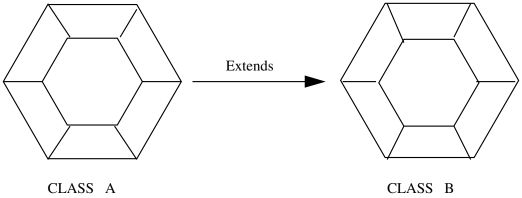
Clearly, containment is a special case of use (i.e., see Item 1.).

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. Here, 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 { .... }
```

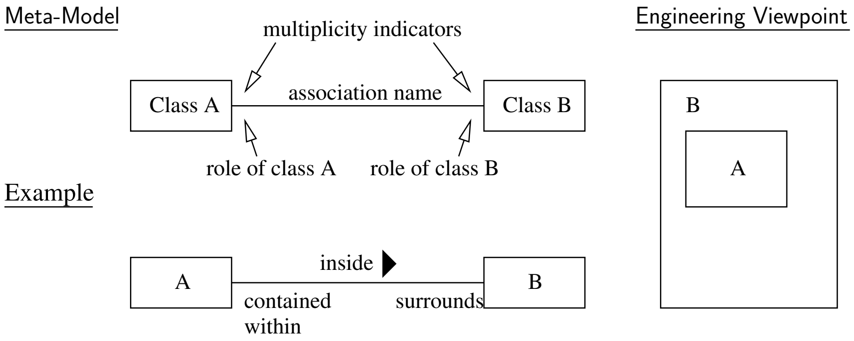
Association Relationships

Definition

As association is a discrete and/or logical relationship between classes. Associations are the glue that tie the elements of a system together.

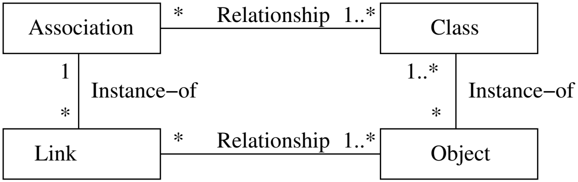
Binary Association Relationships

Binary associations express static bidirectional relationships between two classes.



Binary Association Relationships

Meta-Model for Links and Association Relationships. Links and associations establish relationships among entities within the problem world or the solution world.

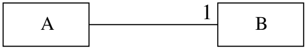
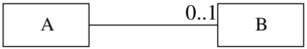
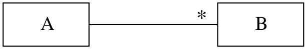
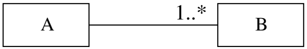
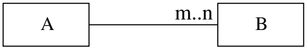


Points to note:

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

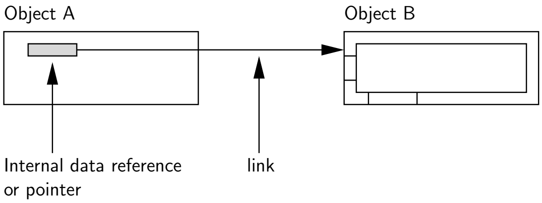
Association Relationships

Multiplicity Constraints. Indicate the number of objects participating in an instance of an association.

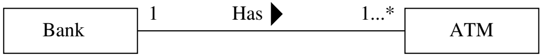
Relationship	Multiplicity
	Exactly one to one
	Optional (zero or one)
	Many (zero or more)
	Many (one or more)
	Numerically specified

Association Relationships

Example 1. Object A links to Object B



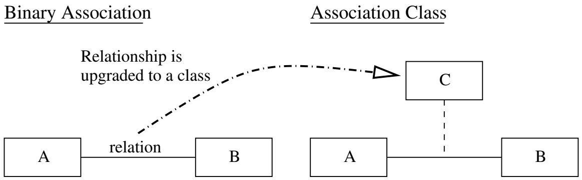
Example 2. A bank and a suite of ATMs



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

Association Class Relationships

From Binary Relations to Association Classes

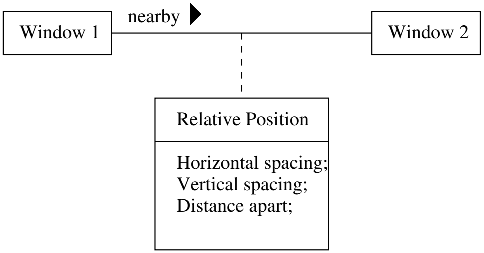


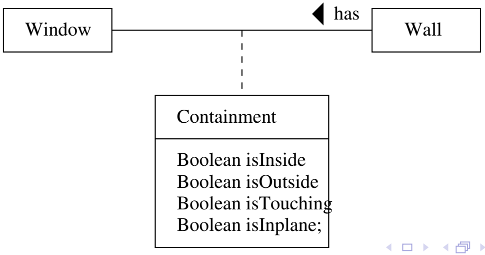
Association classes are used when:

- 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.

Association Class Relationships

Two examples:





Inheritance Mechanisms

Inheritance Mechanisms

Inheritance Structures

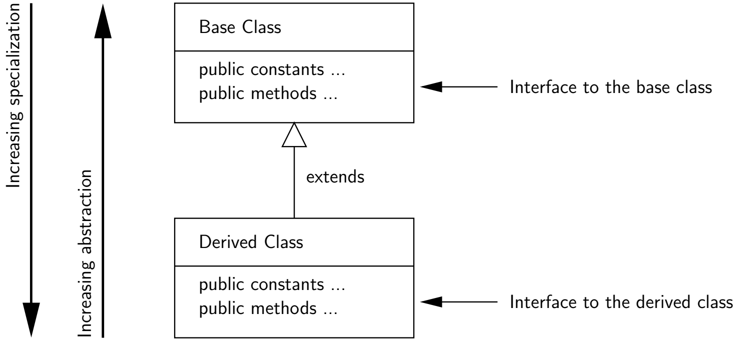
Inheritance structures allow you to capture common characteristics in one model artifact and permit other artifacts to inherit and possibly specialize them. Class hierarchies are explicitly designed for customization through extension.

In this approach to development:

- Forces us to identify and separate the common elements of a system from those aspects that are different/distinct.
- Commonalities are captured in a super-class and inherited and specialized by the sub-classes.
- Inherited features may be overridden with extra features designed to deal with exceptions.

Base and Derived Classes

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



Base and Derived Classes

Points to note:

- 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-level classes.

Inheritance Mechanisms

Example 2. Hierarchy of Temperature Sensors

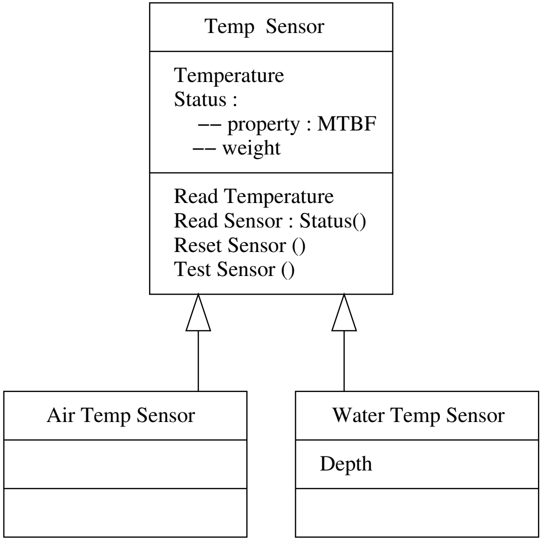
Temperature Thermometer

- 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



Inheritance Mechanisms

Multiple Inheritance Structures

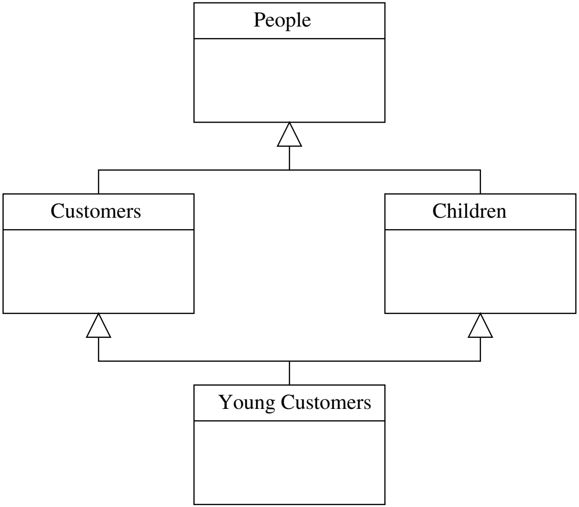
- In a multiple inheritance structure, a class can inherit properties from multiple parents.
- The downside is that properties and/or operations may be partially or fully contradictory.

Example

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

Note. Unlike C++ and Python, Java explicitly prevents multiple inheritance. Java classes can, however, have multiple interfaces.

Inheritance Mechanisms



Example 3. Extending Circle to Colored Circle

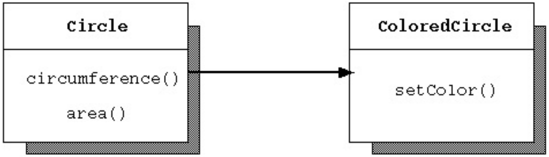
```
/*
 * ===================================================================
 * ColoredCircle(): Implementation of the ColoredCircle class where
 *                  data and circle properties can only be accessed
 *                  through an interface.
 *
 * Written By: Mark Austin                                  April 2019
 * ===================================================================
 */

package objects;

import java.awt.Color;

public class ColoredCircle extends Circle {
    private Color color;

    // Constructor methods

    public ColoredCircle() {
        super();
        this.color = Color.blue;
    }

    public ColoredCircle( double dX, double dY, double dRadius, Color color ) {
        super();

        this.dX = dX;
```

Example 3. Extending Circle to Colored Circle

```
          this.dX = dX;
          this.dY = dY;
          this.dRadius = dRadius;
          this.color   = color;
      }

      // Set and retrieve colors ....

      public void setColor( Color color ) {
         this.color = color;
      }

      public String getColors() {
         return "Color (r,g,b) = (" + color.getRed() + "," + color.getGreen() + "," + colo
      }

      // ===============================================
      // Exercise methods in class ColoredCircle()......
      // ===============================================

      public static void main( String [] args ) {

         System.out.println("Exercise methods in class ColoredCircle");
         System.out.println("=======================================");

         // Create, initialize, and print circle "cA" ...

         ColoredCircle cA = new ColoredCircle( 1.0, 2.0, 3.0, Color.blue );
```

Example 3. Extending Circle to Colored Circle

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()`].