Abstract Classes and Interfaces

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688R, Spring Semester 2023

March 2, 2023

Overview

1. Quick Review
2. Framework for Component-based Design
3. Abstract Classes

Part 1

4. Working with Interfaces
5. Farm Worker Source Code

6. Five Applications
   - Two Factories making Widgets
   - Parsing and Evaluation of Functions with JEval
   - Using Interfaces in Spreadsheets
   - Horstmann's Simple Graph Editor
   - Architecture for Block Interconnect System

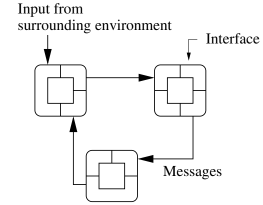
Quick Review

Quick Review: Objects and Classes

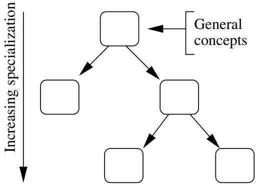
Motivating Ideas

- Simplify the way we view the real world,
- Provide mechanisms for assembly of complex systems.
- Provide mechanisms for handling systems that are subject to change.

Organizational and Efficiency Mechanisms



Network of Communicating Objects



Problem Domain Concepts organized
into a Class Hierarchy.

Quick Review: Object-based Software

Basic Assumptions

- Everything is an object.
- New kinds of objects can be created by making a package containing other existing objects.
- Objects have relationships with other types of objects.
- Objects have type.
- Object communicate via message passing – all objects of the same type can receive and send the same kinds of messages.
- Objects can have executable behavior.
- Objects can be design to respond to occurrences and events.
- Systems will be created through a composition (assembly) of objects.

Quick Reiew: Objects and Classes

Working with Objects and Classes:

- Collections of objects share similar traits (e.g., data, structure, behavior).
- Collections of objects will form relationships with other collections of objects.

Definition of a Class

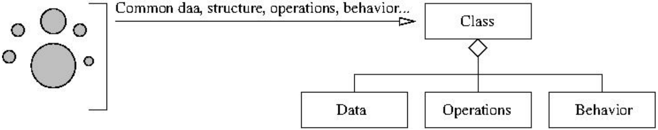
A class is a specification (or blueprint) of an object's structure and behavior.

Definition of an Object

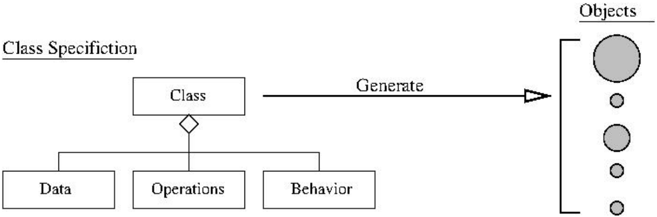
An object is an instance of a class.

Quick Review: Objects and Classes

From Collections of Objects to Classes:



Generation of Objects from Class Specifications:



Quick Review: Objects and Classes

Key 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.

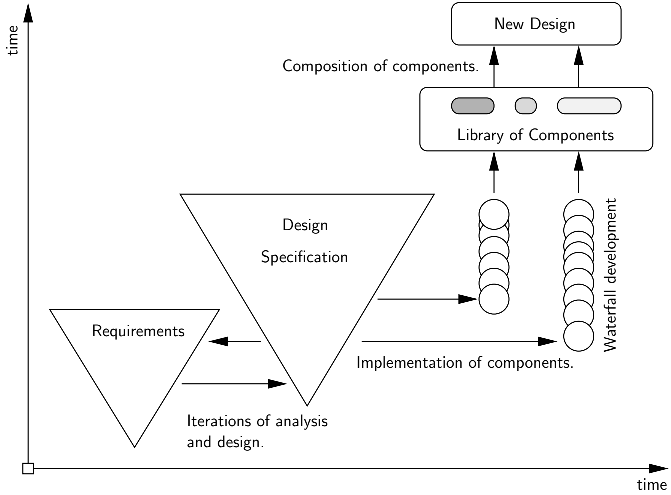
Implicit Assumptions → Connection to Data Mining

- Manual synthesis of the object model is realistic for systems that have a modest number of elements and relationships.
- As the dimensionality of the problem increases some form of automation will be needed to discover elements and relationships.

Framework for Component-based Design

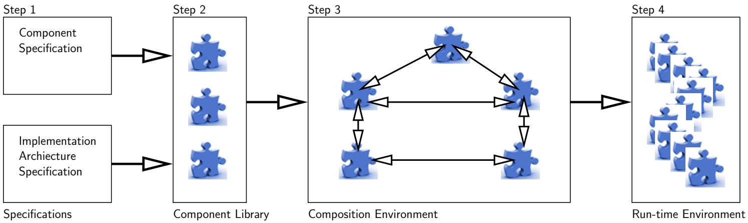
Framework for Component-based Design

Development for Reuse-Focused Design



Framework for Component-based Design

Simplified View of a Component Technology Supply Chain

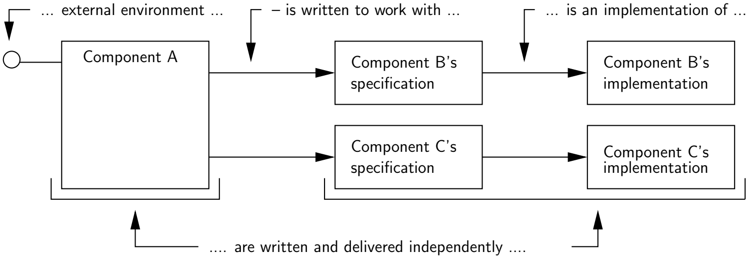


Implementation Requires

- Techniques for describing the overall system architecture.
- Definition of pieces in a way that facilitates assembly with other pieces (e.g., lego blocks).

Framework for Component-based Design

Simple Component-based Software System


... external environment ...
– is written to work with ...
... is an implementation of ...
Component A
Component B's specification
Component B's implementation
Component C's specification
Component C's implementation
.... are written and delivered independently ....

Components B and C are defined via their specifications/interfaces. Component A employs the services of components B and C.

From Component- to Interface-based Design

During the early stages of design where the focus is on understanding the roles and responsibilities of components within a domain, ...

Interface-based Design

Interfaces are a specification for what an implementation should look like.

Benefits:

- Experience indicates that a focus on interfaces as a key design abstraction leads to designs with enhanced flexibility.
- Interface-based design procedures are particularly important for the design and managed evolution of systems-of-systems – e.g., cities.

Abstract Classes

Working with Abstract Classes

Abstract Classes

Abstract classes provide an abstract view of a real-world entity or concept. They are an ideal mechanism when you want to create something for objects that are closely related in a hierarchy.

Implementation

- An abstract class is a class that is declared abstract. It may or may not include abstract methods.
- You cannot create an object from an abstract class – but they can be sub-classed.
- The subclasses will usually provide implementations for all of the abstract methods in its parent class.

Working with Abstract Classes

Example 1. Efficient Modeling of Shapes

A shape is a

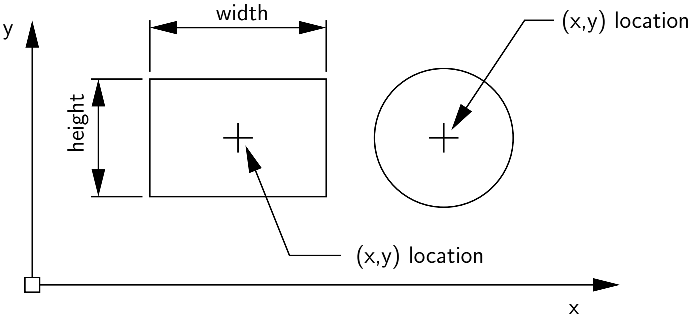
- High-level geometric concept that can be specialized into specific and well-known two-dimensional geometric entities.
- Examples: ovals, circles, rectangles, triangles, octogons, and so forth.

Capturing Shape Data

- There are sets of data values (e.g., vertex coordinates) and computable properties (e.g., area and perimeter) that are common to all shapes.

Working with Abstract Classes

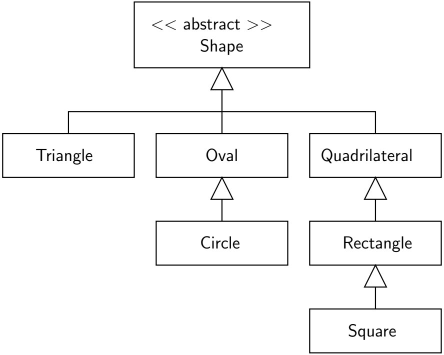
Capturing Shape Data



Computable properties: all shapes have an area, perimeter, an (x,y) centroid and a position or (x,y) location.

Working with Abstract Classes

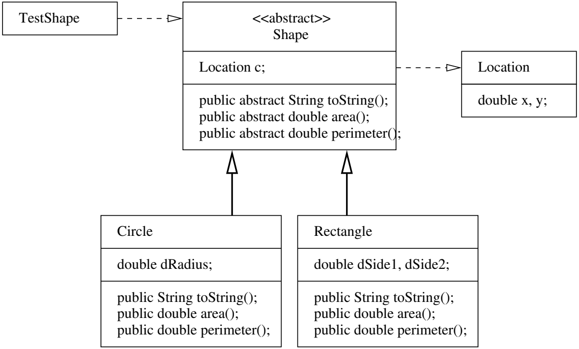
Organizing Shapes into a Natural Hierarchy



Squares are a specific type of rectangle, which, in turn, are a specific type of quadralateral. Circles are a special type of oval.

Working with Abstract Classes

Class Diagram for TestShape Program


TestShape
<<abstract>>
Shape
Location c;
public abstract String toString();
public abstract double area();
public abstract double perimeter();
Location
double x, y;
Circle
double dRadius;
public String toString();
public double area();
public double perimeter();
Rectangle
double dSide1, dSide2;
public String toString();
public double area();
public double perimeter();

All extensions of Shape will need to provide implementations for the methods area(), perimeter() and toString().

Working with Abstract Classes

Implementation Efficiency and Convenience

- Instead of solving problems with algorithms that work with specific object types, algorithms can be developed for shapes.

```
    Shape s[] = new Shape [3] ;

    s[0] = new Rectangle( 3.0, 3.0, 2.0, 2.0 );
    s[1] = new Circle( 1.0, 2.0, 2.0 );
    s[2] = new Rectangle( 2.5, 2.5, 2.0, 2.0 );
```

  The JVM will figure out the appropriate object type at run time.

- The abstract shape class reduces the number of dependencies in the program architecture, making it ammenable to change – trivial matter to add Triangles to the class hierarchy.

Working with Abstract Classes

Walking Along an Array of Shapes

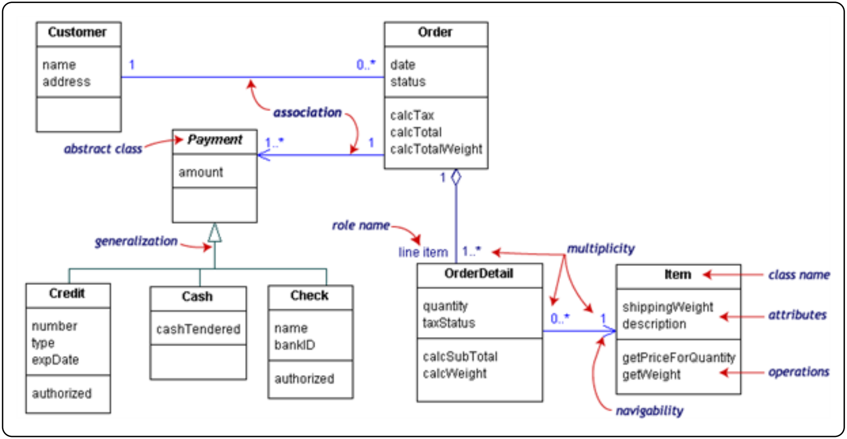
```
System.out.println("---------------------");
for (int ii = 1; ii <= s.length; ii = ii + 1) {
    System.out.println(  s[ii-1].toString() );
    System.out.println( "Perimeter = " + s[ii-1].perimeter() );
    System.out.println("---------------------");
}
```

Program Output:

```
---------------------
Rectangle : Side1 = 3.0 Side2 = 3.0
Perimeter = 12.0
---------------------
Circle : Radius = 1.0 [x,y] = [2.0,2.0]
Perimeter = 6.283185307179586
---------------------
Rectangle : Side1 = 2.5 Side2 = 2.5
Perimeter = 10.0
---------------------
```

Working with Abstract Classes

Example 2. Class Diagram for Operation of a Retail Catalog



Working with Abstract Classes

Points to Note:

- The central class is the Order.
- Associated with each order are the Customer making the purchase and the Payment.
- Payments is an abstract generalization for: Cash, Check, or Credit.
- The order contains OrderDetails (line items), each with its associated Item.

Also note:

- Names of abstract classes, such as Payment, are in italics.
- Relationships between classes are the connecting links.