Inheritance and Composition

COSC 122 Programming II

Rick Halterman
Valley View University
June 2012

Class Reuse

- Classes can be reused in one of two ways:
  - Inheritance
  - Composition
- Here we mean more than just creating instances

Design Issues

Include:

- What attributes must a class possess
- What services must a class provide
- How one class relates to another class
- How one object collaborates with another object

The Unified Modeling Language

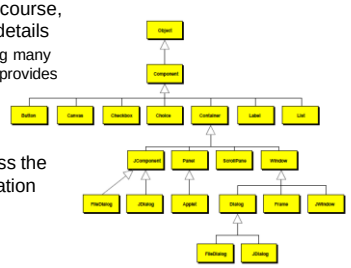
- The UML is used to represent class designs
- The UML is a comprehensive modeling language
- We'll only use a small subset

Why UML?

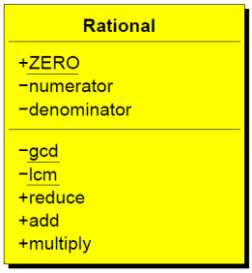
- The UML emerged in the mid-1990s as the dominant graphical modeling language.
- It is a result of combining three major modeling languages:
  - Booch method for OO analysis and design
  - Rumbaugh's Object Modeling Technique (OMT) for OO analysis and design
  - Jacobson's Object-Oriented Software Engineering method (OOSE) emphasizing system requirements
- The UML is quickly becoming the de facto standard OO modeling language

UML is Graphical

- Source code, of course, provides all the details
  - When considering many design issues, it provides too much detail
- The UML, being graphical, can succinctly express the essential information about a design

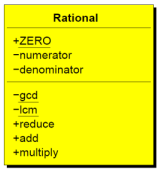


The Rational Class



UML Notation

- The rectangle represents a class.
- The class is divided into three parts with two horizontal lines:
  - The name of the class
  - The attributes (fields) of the class
  - The operations (methods) of the class
- The minus (-) prefix means private The plus (+) prefix means public and thus part of the class interface.
- Class variables and class methods are underlined; instance variables and instance methods are not underlined.



Subclassing

Given an existing class we can derive a brand new class that inherits everything from the original class. We can add functionality to this new class to make this new class different from the original class.

```
public class NewRational extends Rational {
    private double value;
    public NewRational(int n, int d) {
        super(n, d); // Defer work to the superclass constructor
        value = (double) n / d; // Add new code
    }
    public NewRational() {
        super(); // Defer work to the superclass constructor
        value = 0.0; // Add new code
    }
    public double doubleValue() {
        return value;
    }
    // Return the inverse of this rational number
    public Rational invert() {
        return new NewRational(getDenominator(), getNumerator());
    }
    // Computes the quotient of this rational number and fract
    public Rational divide(NewRational fract) {
        return multiply(fract.invert());
    }
}
```

`NewRational` in UML