Technical University of Cluj-Napoca, Romania
Department of Computer Science

Programming Techniques
Revision

Agenda

- Object Oriented Programming Basics
- UML Basics
- Polynomials

Object Oriented Programming Basics
OOP Principles

- **Abstraction**
  - Identify the common features and behavior of a set of objects and represent them in a model (i.e. class)
- **Encapsulation**
  - Group the features and behaviors in an abstract data type and define access levels for an object's data
- **Inheritance**
  - Define and create specialized classes from already defined general classes – the specialized classes can share and extend their behavior without redefining the same behavior
- **Polymorphism**
  - The objects from a class hierarchy can use methods with the same name but with different behavior

Object Oriented Programming Basics
Classes

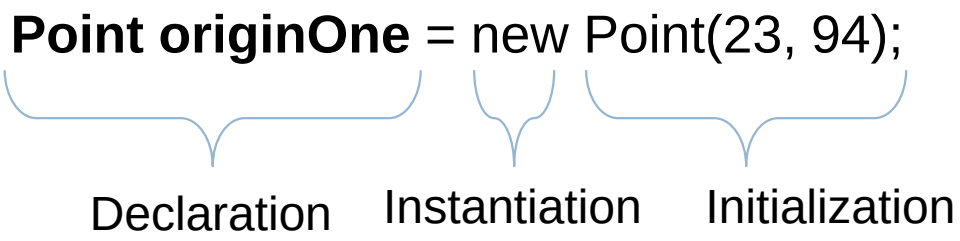
- **Definition**
  - Reference types defined by the user
- **Access modifiers**
  - Public, package private (no modifier specified)
- **Class variables (i.e. fields/attributes/instance variables)**
  - Define the state of an object
  - Access modifiers: public, protected, package-private (no modifier specified), private
- **Class Methods**
  - Define the behavior exposed by the class
  - Constructor – initialize new objects
  - Access modifiers: public, protected, package-private (no modifier specified), private

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Object Oriented Programming Basics
Objects

- **Definition**
  - Instantiation of a class
- **Declaration, Instantiation and Initialization**



- **The “this” keyword**
  - Within an instance method or a constructor, it is a reference to the current object - the object whose method or constructor is being called

Object Oriented Programming Basics
Interfaces

- **Definition**
  - Reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types
    - Method bodies exist only for
      - Default methods (annotated with default keyword) and
      - Static methods - method that is associated with the class in which it is defined rather than with any object
    - Interfaces cannot be instantiated - they can only be implemented by classes or extended by other interfaces.
- A class that implements an interface must implement all the methods declared in the interface
- An interface name can be used anywhere a type can be used

Object Oriented Programming Basics
Abstract Classes

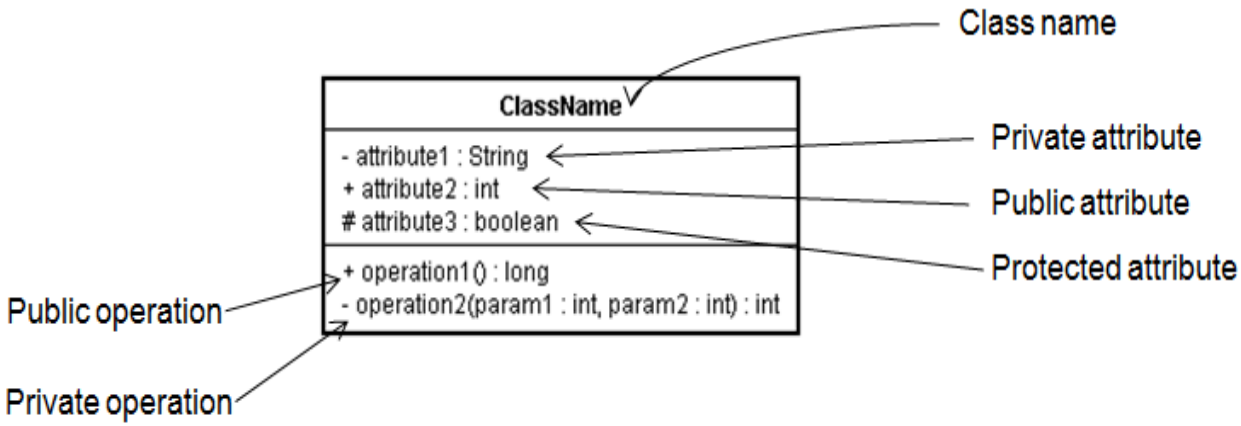
- **Definition**
  - A class that is declared abstract - it may or may not include abstract methods
    - Abstract method - method that is declared without an implementation (without braces, and followed by a semicolon)
  - Abstract classes cannot be instantiated, but they can be sub-classed
- When an abstract class is sub-classed, the subclass usually provides implementations for all of the abstract methods in its parent class
  - If it does not, then the subclass must also be declared abstract

Object Oriented Programming Basics
Inheritance

- **Definition**
  - Process by which a class (**subclass/derived class/extended class/child class**) is derived from another class **(superclass/base class/parent class),** thus reusing the fields and methods of the superclass without having to write them again in the subclass.

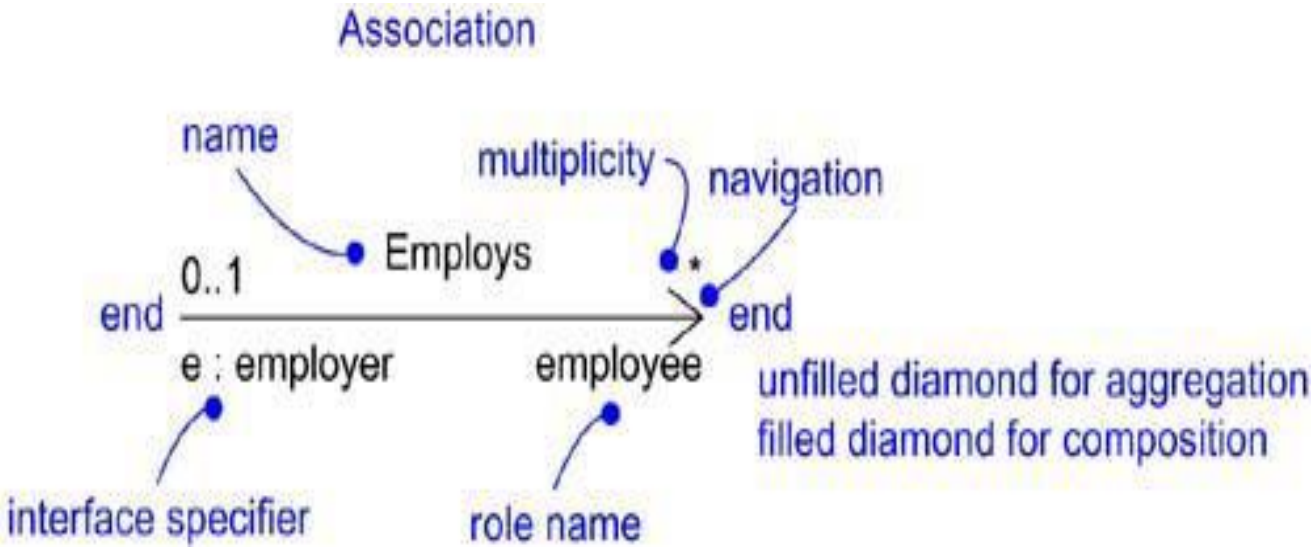
UML Basics
Class Diagrams (I)

- **UML Class Notation**



UML Basics
Class Diagrams (II)

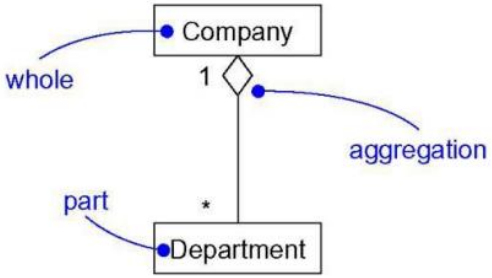
- **UML Class Relationships**
  - **Association -** objects of one thing are connected to objects of another thing
    - An association can have a name used to describe the nature of the relationship
    - When a class participates in an association, it has a specific role that it plays in that relationship



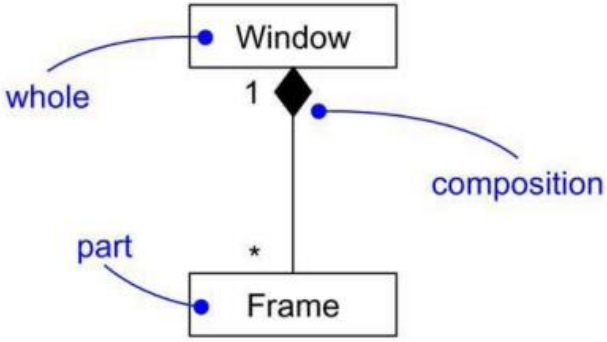
UML Basics
Class Diagrams (III)

- **UML Class Relationships**
  - **Association –** special cases

Aggregation

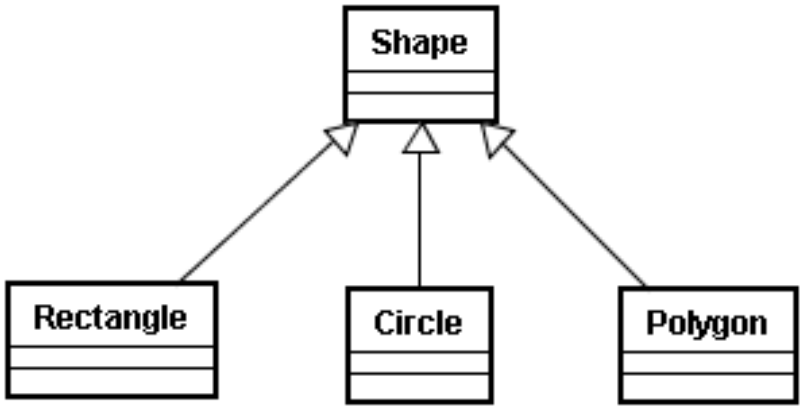


Composition



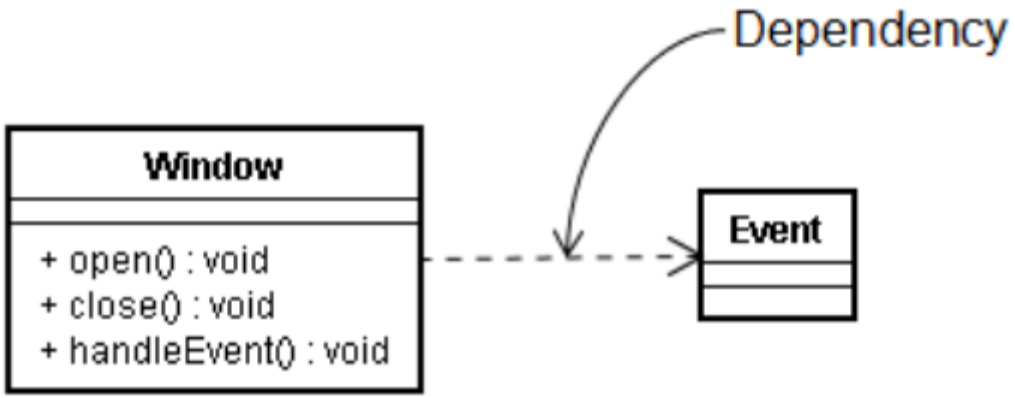
UML Basics
Class Diagrams (IV)

- **UML Class Relationships**
  - **Generalization -** established between a general kind of thing (superclass) and a more specific kind of thing (subclass) => the child is substitutable for a declaration of the parent



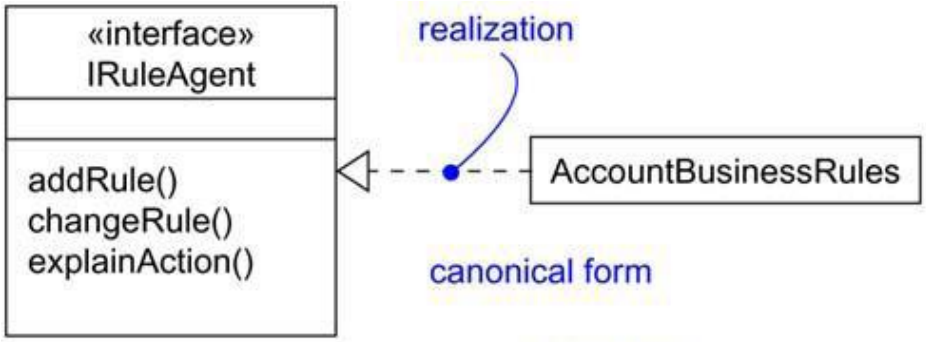
UML Basics
Class Diagrams (V)

- **UML Class Relationships**
  - **Dependency -** shows that one class uses operations from another class or it uses variables or arguments typed by the other class (if the used class changes, the operation of the other class may be affected)



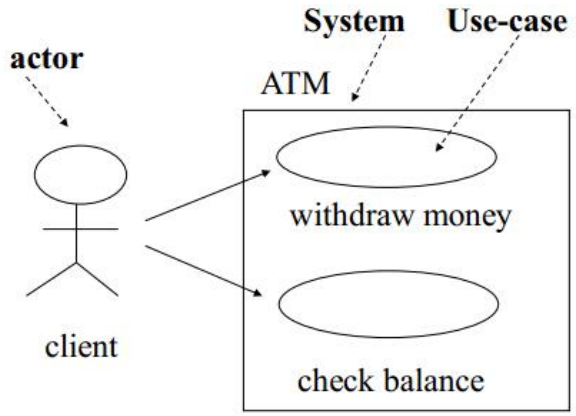
UML Basics
Class Diagrams (VI)

- **UML Class Relationships**
  - **Realization -** semantic relationship between classifiers (e.g classes, interfaces, collaborations use cases) in which one classifier specifies a contract that another classifier guarantees to carry out



UML Basics
Use Case Diagrams (I)

- **Use case** - description of a set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor
- **Actor** - set of roles that users of use cases play when interacting with these use cases
  - Typically, an actor represents a role that a human or hardware device or even another system plays with a system



***Use case**: <use case goal>*
***Primary actor**: <a role name for the actor who initiates the use case>*
***Main success scenario**: <the steps of the main success scenario from trigger to goal delivery>*
***Extensions**: <alternate scenarios of success or failure>*

UML Basics
Use Case Diagrams (II)

- **Use Case**: Buy a product
- **Primary Actor**: Customer
- **Main Success Scenario**:
  - Customer browses catalogue and selects item to buy.
  - Customer goes to the check out
  - Customer fills in shipping information
  - System presents full pricing information
  - Customer fills in credit card information
  - System authorizes purchase
  - System confirms sale and sends confirming email to customer
- **Extensions**:
  - 6a: System fails to authorize credit purchases -> Customer may reenter credit card information or may cancel

(From http://www.cems.uwe.ac.uk/~jsa/UMLJavaShortCourse09/CGOutput/Unit1/unit1%280809%29/page_23.htm)

Polynomials
Definition

$$P(X) = c_n * X^n + c_{n-1} * X^{n-1} + c_{n-2} * X^{n-2} + \ldots + c_1 * X + c_0$$

- c_1, c_2, …, c_n – coefficients
- n – polynomial degree
- X – variable (indeterminate)

Polynomials
Arithmetic of Polynomials (I)

$$P_1(X) = 4X^5 - 3X^4 + X^2 - 8X + 1$$

$$P_2(X) = 3X^4 - X^3 + X^2 + 2X - 1$$

Addition

$$(P_1 + P_2)(X) = 4X^5 - 3X^4 + X^2 - 8X + 1 + 3X^4 - X^3 + X^2 + 2X - 1 =$$
$$= 4X^5 - X^3 + 2X^2 - 6X.$$

Subtraction

$$(P_1 - P_2)(X) = 4X^5 - 3X^4 + X^2 - 8X + 1 - 3X^4 + X^3 - X^2 - 2X + 1 =$$
$$= 4X^5 - 6X^4 + X^3 - 10X + 2.$$

Polynomials
Arithmetic of Polynomials (II)

$$P_1(X) = 3X^2 - X + 1$$

$$P_2(X) = X - 2$$

$$P_1(X) = X^3 - 2X^2 + 6X - 5$$

$$P_2(X) = X^2 - 1.$$

Multiplication

$$\begin{array}{l} (3X^2 - X + 1) \cdot (X - 2) \\ \underline{3X^3 - X^2 + X} \\ \qquad -6X^2 + 2X - 2 \\ \hline 3X^3 - 7X^2 + 3X - 2 \end{array}$$

Division

$$\begin{array}{l} (X^3 - 2X^2 + 6X - 5) : (X^2 - 1) = X - 2 \\ \underline{-X^3 \qquad + X} \\ \qquad -2X^2 + 7X - 5 \\ \qquad \underline{2X^2 \qquad - 2} \\ \qquad\qquad 7X - 7 \end{array}$$

Polynomials
Arithmetic of Polynomials (III)

Value of a polynomial

$$P_1(X) = 3X^2 - X + 1$$

$$P_1(2) = 3 \cdot 2^2 - 2 + 1 = 3 \cdot 4 - 2 + 1 = 12 - 2 + 1 = 11$$

Assignment 1

- **“Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.”**
  - Tasks:
    - 1. Create the conceptual class diagram for assignment 1.
    - 2. Create a graphical user interface for assignment 1 with mockup methods.

Tasks

Bibliography

- https://cnamd09.wikispaces.com/file/view/0914+Polinoame.pdf