Object-Oriented Modeling Using UML

CS151

Chris Pollett

Aug. 29, 2005.

Outline

- Objects and Classes
- Modeling Relationships and Structures

Some Terms and Concepts

- Objects and classes are fundamental to OO development.
- They both can be viewed from a real world perspective and from within the OO-model.

	Interpretation in Real World	Representation in the Model
Object	An object represents anything in the real world that can be distinctly identified	An object has an identity, a state and a behavior.
Class	A class represents a set of objects with similar characteristics and behaviors.	A class characterizes the structure of states and behaviors that are shared by all objects

More Terms and Concepts

- An *identity* distinguishes one object from another.
- An object (aka instance) consists of a set of *fields* (aka attributes).
- Each field has a type and a value. These give the object a state
- The behavior of on an object is defined by a set of *methods* (aka member functions/operations) that may operate on it.
- State + Methods = *Features*

Still More Concepts

- Two objects are *equal* if their states are equal. Two objects are identical if they are the same object.
- *Accessors* are methods which do not modify the state of an object.
- *Mutators* are methods which do modify the state of an object.
- An object is mutable/immutable depending on whether its state can be changed.

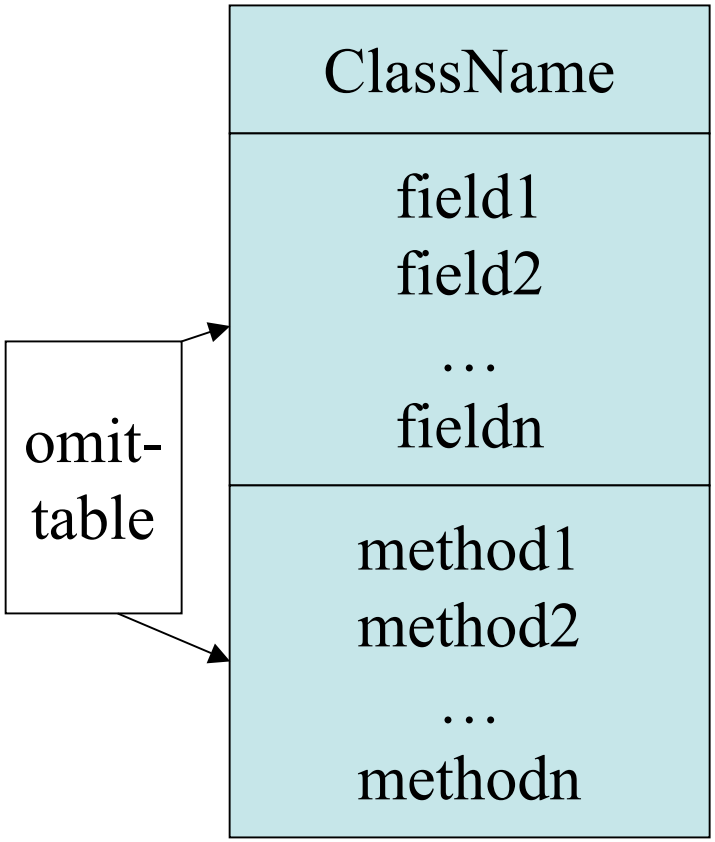
An example class:

```
class Point
{       int x, y; //fields
        public void move(int dx, int dy) {/*implementation*/} // Method
}
```

UML

- Unified Modeling Language
- Used during the design phase
- We will use this language to model different kinds of OO software project
- Given a UML diagram we can then proceed to actually implement it in some language like Java

UML Notations for Classe



Fields can either be in the form:

[Visibility][Type]Name[[*Multiplicity*]*] [= Value]* Ex. int a

or

[Visibility]Name [[*Multiplicity*]*] : Type [= Value]* Ex. a: int

Methods can either be in the form:

[Visibility][Type] Name ([Param],…)

Ex.private int getDay (Date d)

or

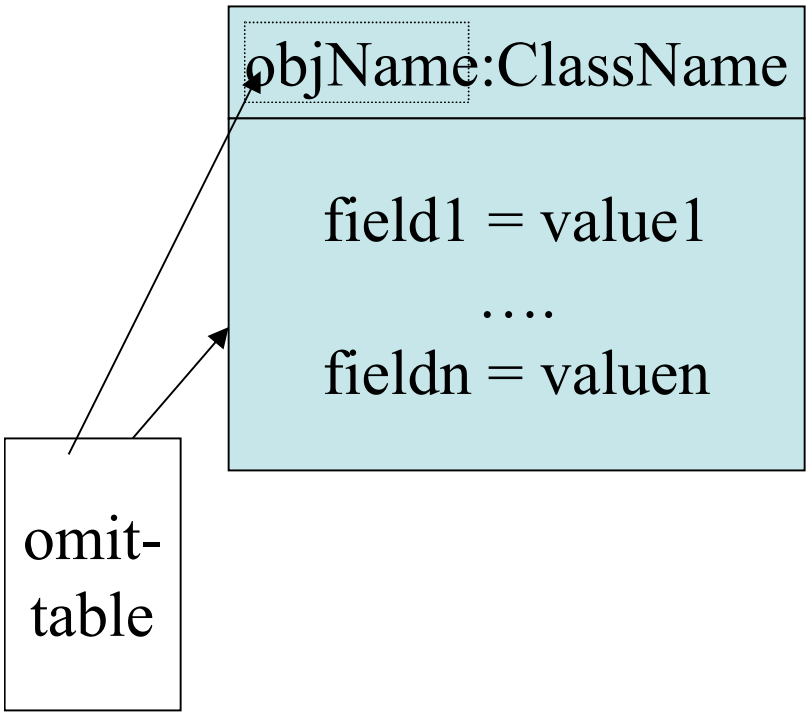
[Visibility] Name ([Param],…) :Type

Ex. -getDate(d:Date) :int

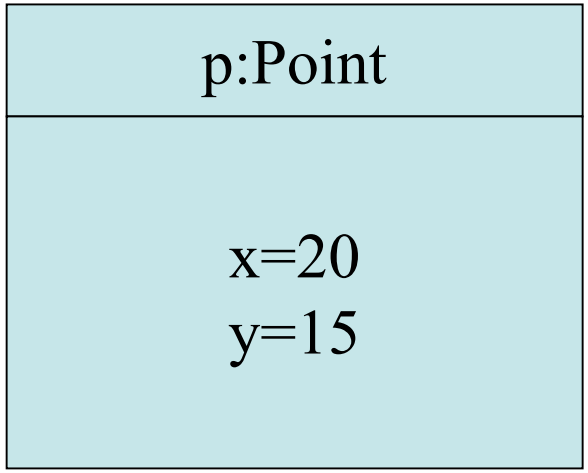
More on Visibility

Visibility	Java Syntax	UML Syntax	Meaning
public	public	+	any class can see
protected	protected	#	same package and subclasses can see
package		~	package can see
private	private	-	class only can see

UML Notation for Objects



For example,



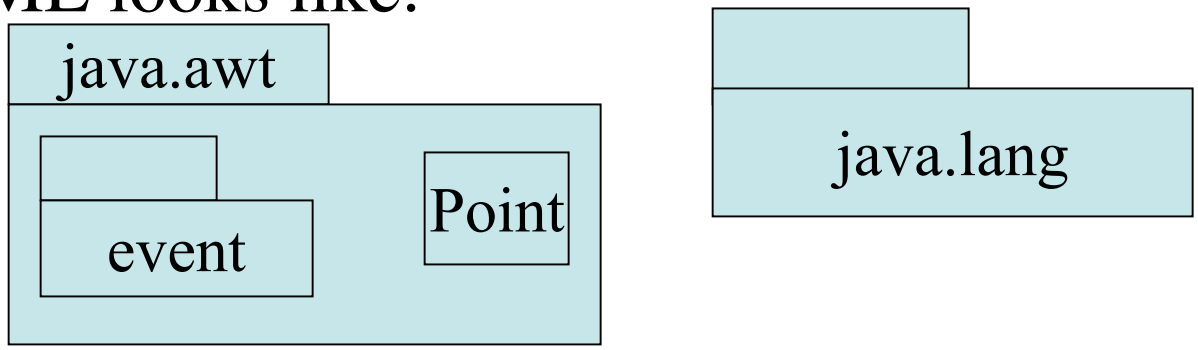
Message Passing

- Objects communicate with each other by means of *message passing*.
- A message represents a command sent to an object -- known as the recipient of the command -- to perform an action (invoke one of its methods).
- A message consists of a receiving object, a method to invoke, and any arguments for this method.

```
p1.move(10,20); /* recipient is p1, method is move,
    arguments are (10,20) */
```

UML Notations of Packages

- Classes are often grouped together into packages.
- We'll follow the convention that packages should have all lower case names. Ex edu.sjsu.cs.pollett
- Using internet domains (convention in reverse order) ensures packages names will be unique.
- UML looks like:



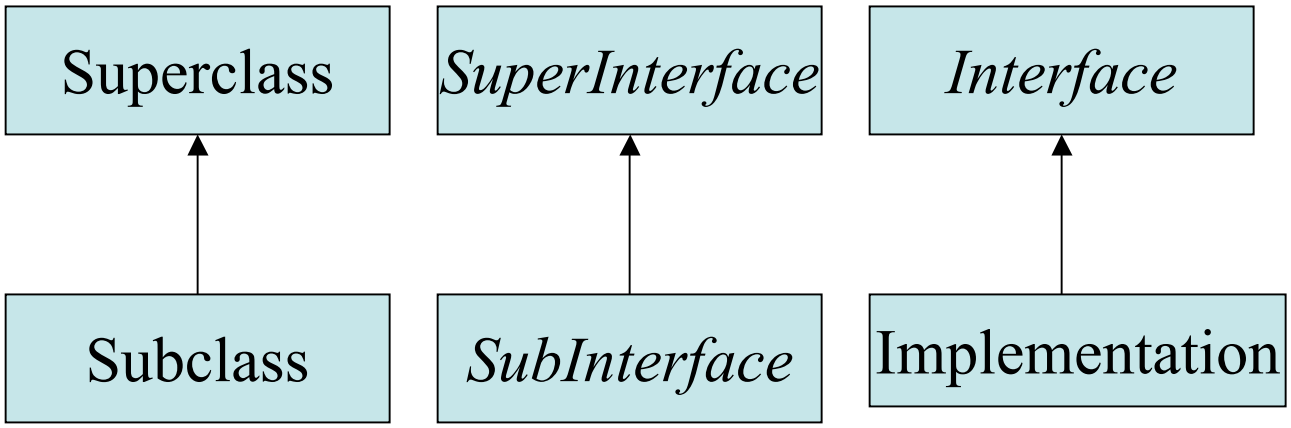
OO Principles

- Modularity -- a complex software system should be decomposed into a set of highly cohesive but loosely couple modules
- Abstraction -- functionalities of a module should be characterized in a succinct and precise description known as a *contractual interface*.
- Encapsulation - implementation of a module should be separated from the contractual interface and hidden from the module user
- Polymorphism - different service providers can implement the same contractual interface

Modeling Relationships

- A UML class diagram consists of a set of nodes to represents classes or interfaces and a set of links to represent relationships between these classes.
- Possible relationships:
  - Inheritance -- includes extension and implementation
  - Association -- includes aggregation and composition
  - Dependency

UML Notation for Inheritance



Extension of a class

Extension of an interface

Implementation of an interface

Levels of Abstraction

- Abstraction can be ordered into more than two levels.
- The higher the level the more general the abstraction