Model-Oriented Programming with Umple

Gunter Mussbacher
ECE, McGill University, Canada ◄► gunter.mussbacher@mcgill.ca

Based on material from: Timothy C. Lethbridge, University of Ottawa

Table of Contents

- Philosophy of Model-Oriented Programming (MOP)
- Umple
  - Introduction
  - Attributes
  - Associations
  - Generalization
  - Patterns
  - Aspects and Mix-ins
  - Key Advantages

Umple Online … links to online code examples

Philosophy of Model-Oriented Programming

- Modeling abstractions are embedded directly in programming languages
  - E.g., UML associations, attributes, and state machines (state machines not covered today)
- Programs and models are unified
  - Traditional models can be expressed as program code
  - Traditional code is really just modeling at a more detailed level (a lower level of abstraction)
- The programmer/modeler has a choice of workflow
  - Model-first: use just the modeling notations, then add detail
  - Incremental re-engineering: take existing code and incrementally convert it to use modeling abstractions

Philosophy of Model-Oriented Programming

- Text-diagram duality
  - The abstractions in the model-oriented programming language can be rendered directly as a diagram (unambiguously, without reverse engineering)
  - The diagram can be edited to update the code (live)
  - The code can be edited to update the diagram (live)
- The model/code can be compiled to build a complete system
  - No editing needed of code generated from the model since all needed algorithms, methods, etc. are present in the model/code source
  - No 'round tripping'

Umple: A MOP Technology and Language Family

- Adds associations, attributes, patterns, and state machines to programming languages
  - Java, PHP, Ruby, C++
- Stand-along code-generator and diagram/text editor is online at
  - http://cruise.site.uottawa.ca/umpleonline/
  - Limited to a single file, but incorporates many examples
  - Code generation to the above languages and modeling environments such as EMF and Papyrus
- Works from the command line, with Eclipse, and with other tools for diagram generation and code generation
  - Xtext
  - EMF's Ecore
  - Papyrus open-source modeling

Umple: What's in the name?

- UML Programming Language
- Ample
  - All you need to merge modeling and programming
- Simple
  - Easy for programmers or modelers to adopt, without a significant learning curve
  - Easy to convert existing code
- Umple is written in Umple!
- Umple is a modeling tool that is developed in a fully model-driven manner!

Umple Classes and Attributes

Umple Online

```
class Student
{
  studentNumber; // defaults to String
  String grade;
  Integer entryAverage; // implemented as int
}
```

- A UML/Umple attribute is not the same as an instance variable (member variable)
  - Not all instance variables are attributes, some model associations (discussed later)
  - Attributes can have properties like immutability, uniqueness… (discussed later)

Datatypes for Declaration of Attributes

- Umple treats the following attribute types as special
  - `String` (always the default if unspecified)
  - `Integer`
  - `Double`
  - `Boolean`
  - `Date`
  - `Time`
- Code generation from the above will generate suitable types in the underlying language (Java, PHP, etc.)
- Umple classes can be used as types, but consider declaring associations instead (discussed later)

Additional Options for Attributes

- `name;`
  - Set in constructor, getter and setter are generated
- `name = "Unknown";`
  - Initial value set to default, not required in constructor
- `immutable idNumber;`
  - Cannot be changed after being set in constructor
- `lazy name;`
  - A constructor argument is not required (numbers are initialized to zero, Booleans to false, everything else to null)

Additional Options for Attributes

- `lazy immutable name;`
  - Can be set once, right after construction, and is immutable after that
  - Useful for frameworks where objects are created without initializing values
- `unique String ipAddress;`
  - Value must be different in each object
- `autounique flightNumber;`
  - Umple assigns the next available number
- `const Integer MAX = 1000;`
  - Constants (in Java they become static)

Additional Options for Attributes

- `defaulted type = "Long";`
  - If the value is reset, the default is re-established
  - Such attributes can never be ‘unspecified’

- `Integer length;`
- `Integer width;`
- `Integer perimeter = { 2*getLength() + 2*getWidth() }`
- `Integer area = { getLength() * getWidth() }`
  - Derived attributes

- `String[] names;`
- `String[0..3] addressLines;`
  - Multiplicities other than 1

Code Generation from Attributes

- Arbitrary methods written inline in Umple must access the attributes using a defined API

- All attributes become private instance variables
  - User-written code is not allowed to access these

- Constructors arguments are generated where an initial value is needed

- **`public getX()`**
  - Always call this to access the attribute

- **`public setX()`**
  - Available except for **`immutable`**, **`const`**, **`autounique`**, and derived attributes

Umple Associations

Umple Online

```
class Student { id; name; }

class Course { description; code; }

class CourseSection {
  sectionLetter;
  1..* -- 1 Course;  // association declared in a class
}

association {
   * CourseSection -- * Student registrant;
}
```

Two Ways of Writing Associations

```
class A {1 -- * B;}
class B {}
```

- Is semantically identical to

```
class A {}
class B {}
association {1 A -- * B;}
```

API for Manipulating Links of Associations

- Accessing the association end at class A
  - `public B getB(int index)`
  - `public List<B> getBs() /* unmodifiable */`
  - `public int numberOfBs()`
  - `public boolean hasBs()`
  - `public int indexOfB(B aB)`
  - `public B addB() /* creates new B */`
  - `public boolean addB(B aB)`
  - `public boolean removeB(B aB)`

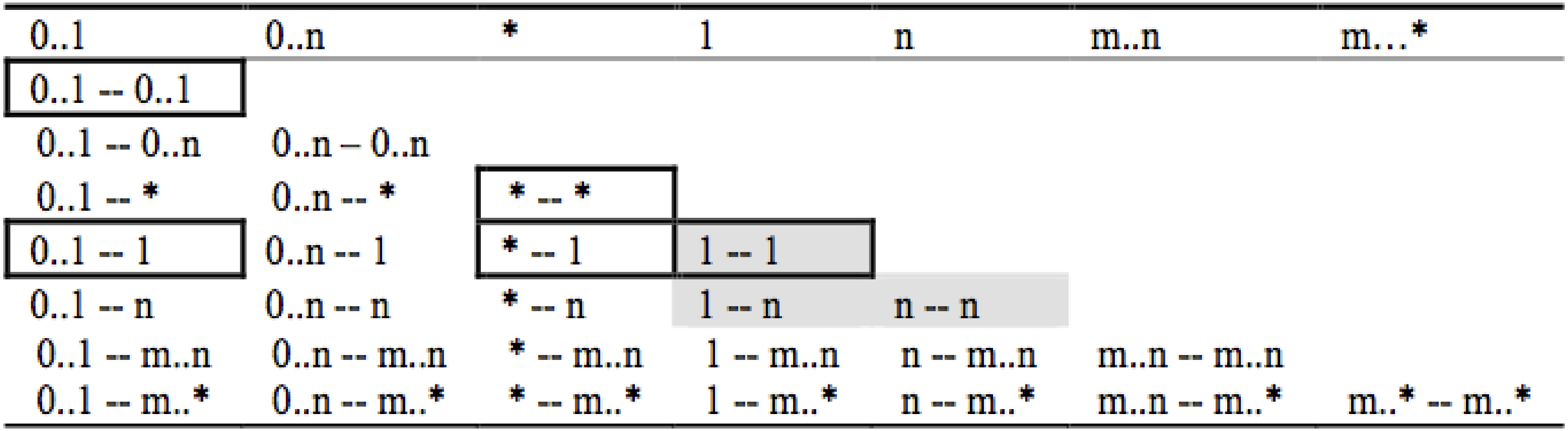
- Acessing the association end at class B
  - `public A getA()`
  - `public boolean setA(A aA)`
  - `public void delete()`

Benefits of Associations at Programming Level

- Saves writing a large amount of ‘boilerplate’ code
  - Savings can be 10:1
- Referential integrity
  - 1 X -- * Y
  - An X points to some Ys; a Y always points to an X
  - Bidirectionality of links managed

Full Support for Associations

- Umple supports the full set of UML associations
  - Directional Associations (m and n can be any number)
    - * -> 0..1, * -> 1, * -> *, * -> m..n, * -> n, * -> m..*, and * -> 0..n
  - Reflexive Associations
    - 0..1, 0..n, *, 1, n, m..n, m..*
  - Bidirectional non-Reflexive Associations
    - The boxed ones are the common cases

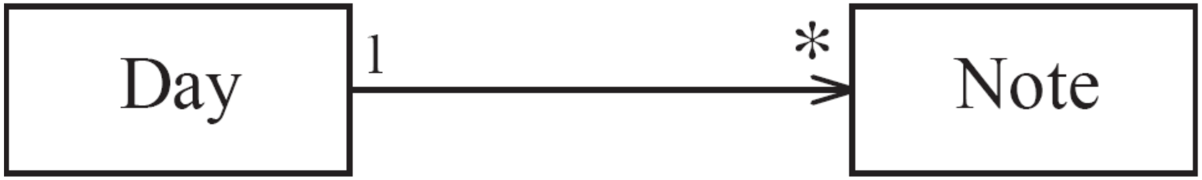


0..1	0..n	*	1	n	m..n	m...*
0..1 -- 0..1						
0..1 -- 0..n	0..n – 0..n					
0..1 -- *	0..n -- *	* -- *				
0..1 -- 1	0..n -- 1	* -- 1	1 -- 1			
0..1 -- n	0..n -- n	* -- n	1 -- n	n -- n		
0..1 -- m..n	0..n -- m..n	* -- m..n	1 -- m..n	n -- m..n	m..n -- m..n	
0..1 -- m..*	0..n -- m..*	* -- m..*	1 -- m..*	n -- m..*	m..n -- m..*	m..* -- m..*

Directional Associations (Navigability)

Umple Online

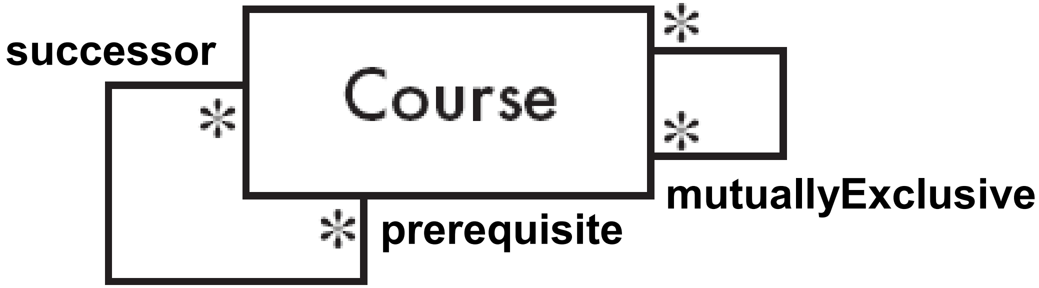
```
class Day {
  1 -> * Note;
}

class Note {}
```



Reflexive Associations

Umple Online

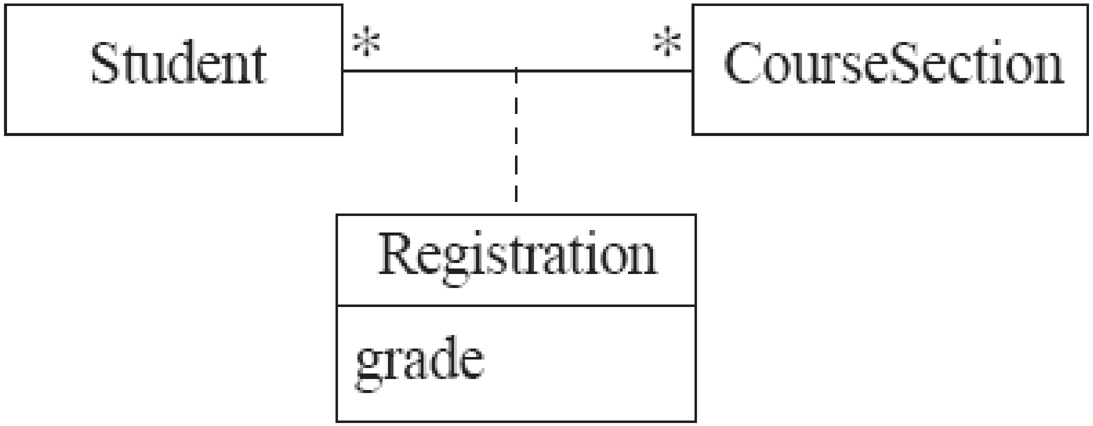
```
class Course {
   * prerequisite -- * Course successor;
   * self mutuallyExclusive;
}
```



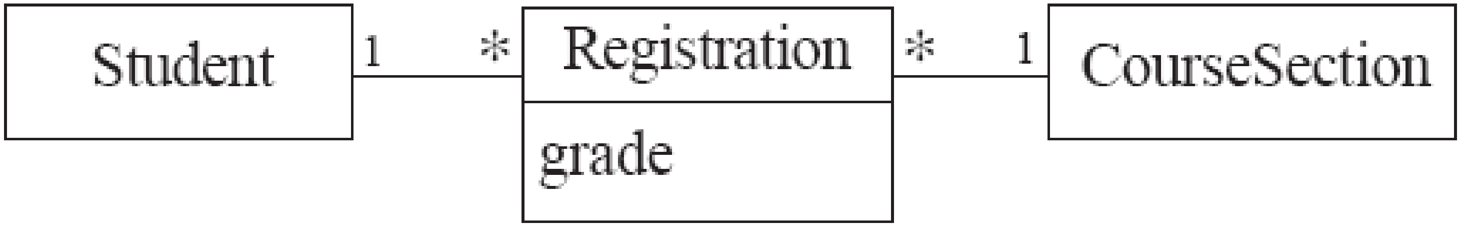
- It is possible for an association to connect a class to itself

Association Classes

Umple Online



Umple Online



Generalization / Inheritance

- Inheritance: the implicit possession by all subclasses of features defined in its superclasses

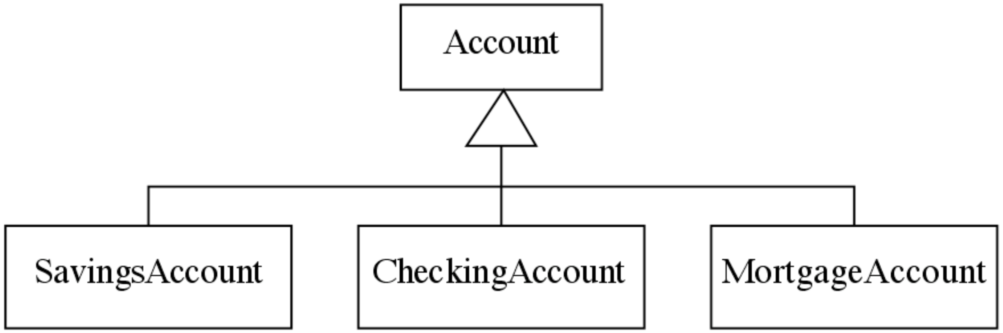
Umple Online

```
class Account {}
class SavingsAccount { isA Account; }
class CheckingAccount { isA Account; }
class MortgageAccount { isA Account; }
```

- The `isA` keyword is used so Umple code is visually distinct from code in other languages (different languages use different notations)

- Alternative notation:

```
class Account {
 class SavingsAccount {}
 class CheckingAccount {}
 class MortgageAccount {}
}
```



Support for Patterns: Singleton

Umple Online

```
class University {
  singleton;
  String name;
}
```

has to be a lazy attribute

- Generated code:

```
private static University theInstance = null;
private University() {
  name = null;
}
public static University getInstance() {
  if(theInstance == null) {
    theInstance = new University();
  }
  return theInstance;
}
```

Before and After Code Injection

- Provides aspect-oriented capabilities

Umple Online

```
class Person {
   name;
   before setName {
      if (aName != null || aName.length() > 20) {
         return false; }
   }
   after setName {
      System.out.println(
         "Successfully set name to : " + aName);
   }
}
```

- Asterisks can be used for pattern matching

Mix-in Capability

- Define features in separate files and merge those features by compiling the classes together

- In one file
  - `class X { Integer a; }`
- In another file
  - `class X { Integer b; }`

- Class X now has two attributes

Key Advantages of Umple and MOP

- Programmers can use Umple as little or as much as they want
  - Pure Java/PHP/Ruby/C++ is just ‘passed through’
  - Learning curve is low, and adoption can be gradual
- Great learning tool to understand the benefits of modeling
- Umple’s code generation is state-of-the art
- Support for multiple programming languages
- Umple home page
  - http://cruise.site.uottawa.ca/umple/