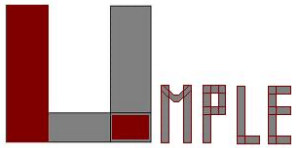
Experiential Learning for Software Engineering Using Agile Modeling in Umple

A Tutorial at CSEE&T 2020

Timothy C. Lethbridge, I.S.P, P.Eng.
University of Ottawa, Canada

Timothy.Lethbridge@ uottawa.ca

http://www.umple.org



Outline

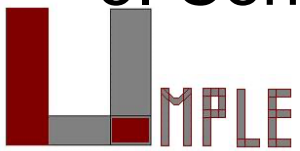
1. **Motivations for better ways of teaching software engineering (Slides 3-9)**

2. Overview of **Umple** (including demonstration of how various concepts can be taught) **(Slides 11-72)**

3. Summary of teaching **introductory software engineering** with Umple (mostly integrated with part 2) **(Slide 73)**
   - Model-driven development
   - Practical modeling in the classroom (examples included)
   - Agility

4. Teaching a **capstone course** using Umple (**Slides 74-77**)

5. Conclusion (**Slide 78**)



Motivations for better teaching of SE #1:
We need to teach agility better

Modern software engineering in industry now widely incorporates agility, including

—Small frequent releases, with continuous integration

—Issue tracking of bugs and feature requirements with

- Branching and merging with pull requests
- Traceability of changes back to issues
- Diff comparisons for commenting, inspection
- Design discussions (forming part of lean documentation)

—Automated testing (even Test-Driven development)

We need projects and methods to teach this

Motivations for better teaching of SE #2:
We need to teach design abstraction better

Greater abstraction of design => Modeling

Key industrial sectors (e.g. telecom, automotive) now perform:

- Advanced modeling with sophisticated tools
- Code generation

But tools available for teaching have some combination of weaknesses:

- Too expensive for educators and small teams
- Excessively complex, so hard to learn/use
- Generate poor/incomplete code
- Don't work well with agility, particularly since non-textual

Motivations for better teaching of SE #3: Students need feedback on working systems

Too much of SE is taught as

- Theory only
  —e.g. just syntax and semantics of modeling notations
- Design without real feedback
- Development of toy systems (typically unrealistic)

Students need to be able to use all SE techniques and get feedback from

- Instant, automated model analysis (like a compiler)
  —Enabling an agile [(fix or redesign) -> retry] loop
- Failure of their system to work or satisfy users/requirements

Research we have done to learn about the state of modeling tools suitable for teaching

Agner, L.T.W., Lethbridge, T.C., and Soares, I.W., **Student experience with software modeling tools**, *Software & Systems Modeling*, Volume 18, Issue 5, 3025-3047 (Springer, Sept 2019)

https://rdcu.be/bfxpo , doi: 10.1007/s10270-018-00709-6

Badreddin, O., Khandoker, R., Forward, A., Masmali, O., Lethbridge, T.C., **A decade of software design and modeling: A survey to uncover trends of the practice**, *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, Copenhagen, Page Range: 245-255, (2018) Conference Date: October 15, 2018

doi: 10.1145/3239372.3239389

Sturm, A., Lethbridge, TC, **Poster: Are Our Students Engaged in Their Studies? Professional Engagement vs. Study Engagement**, *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, Gothenburg, Page Range: 149-150, IEEE, (2018) Conference Date: May 15, 2018

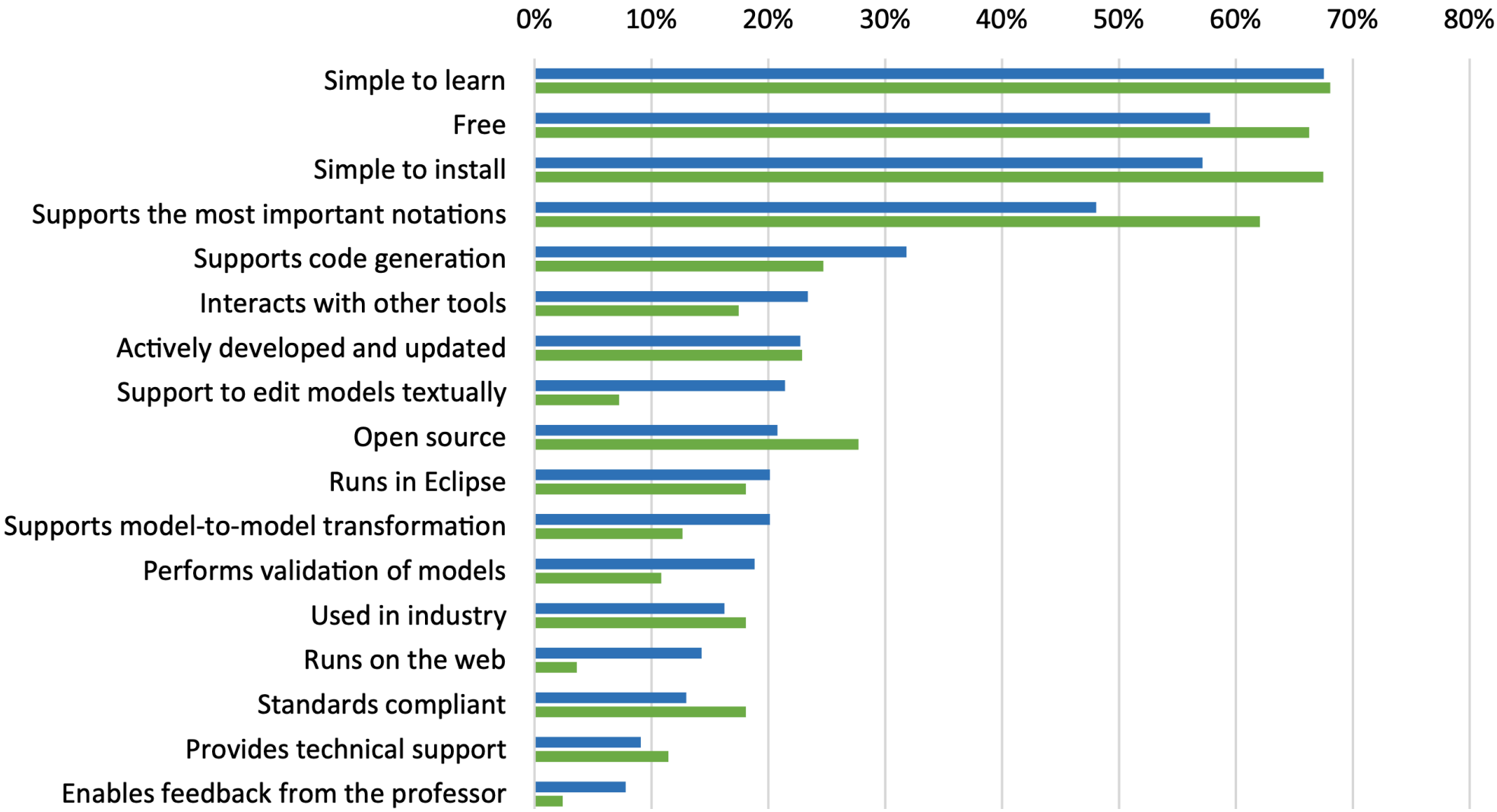
https://ieeexplore.ieee.org/abstract/document/8449474

Agner, Luciane T. W. and Lethbridge, T.C., **A Survey of Tool Use in Modeling Education**, *Models 2017*, Austin, Texas, Page Range: 303-322, IEEE Computer Society, (September 2017) Conference Date: September 2017

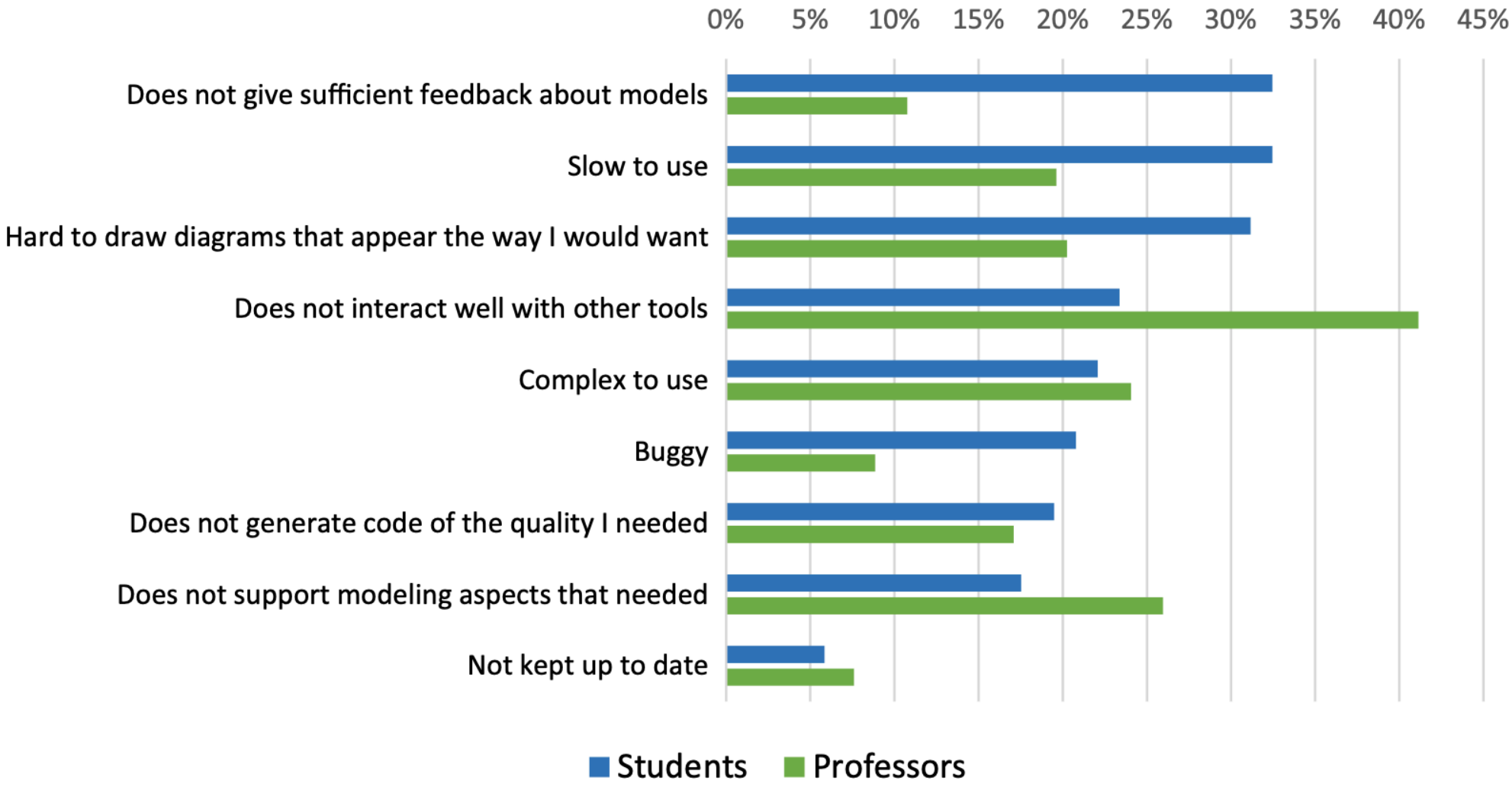
http://ieeexplore.ieee.org/document/8101276/, doi: 10.1109/MODELS.2017.1

+ others: http://publlications.umple.org

From our 2017/18 survey: What students and profs want in modeling tools to help learning



From 2017/18 survey: Key problems in modeling tools



Tool comparison from 2017/18 survey: For Umple, we have fixed slowness + bugs

Tools	Argo UML	Astah	Eclipse M. Tools	Magic Draw	Papyrus	Star UML	Umple	USE	Visual Parad.
Does not give sufficient feedback about models	50.0%	41.7%	40.0%	33.3%	0.0%	31.8%	15.0%	16.7%	25.0%
Hard to draw diagrams the way I want	50.0%	16.7%	20.0%	11.1%	25.0%	45.5%	40.0%	41.7%	25.0%
Slow to use	16.7%	16.7%	50.0%	61.1%	50.0%	22.7%	55.0%	25.0%	25.0%
Does not interact well with other tools	25.0%	25.0%	20.0%	27.8%	25.0%	31.8%	25.0%	16.7%	12.5%
Does not generate code of the quality I needed	25.0%	16.7%	0.0%	16.7%	25.0%	31.8%	15.0%	16.7%	12.5%
Complex to use	16.7%	16.7%	60.0%	16.7%	25.0%	18.2%	5.0%	50.0%	12.5%
Buggy	16.7%	0.0%	20.0%	22.2%	37.5%	22.7%	55.0%	0.0%	12.5%
Does not support modeling aspects that needed	16.7%	16.7%	0.0%	11.1%	12.5%	27.3%	20.0%	25.0%	0.0%
Not kept up to date	8.3%	8.3%	0.0%	5.6%	0.0%	4.5%	15.0%	8.3%	12.5%

Before We Move On:
Tell Me About Yourselves:

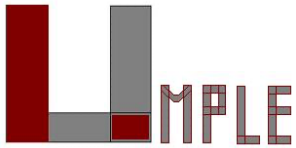
Quick survey about

- What modeling tools do you use?
- Whether you are an academic, industrial practitioner or student
- Whether you generate code

https://www.surveymonkey.ca/r/CSEET-Umple2020

Some lists of modeling tools, as rated by others:

- https://www.guru99.com/best-uml-tools.html
- https://modeling-languages.com/uml-tools/



PART 2: Umple: Simple, Ample, UML Programming Language

Open source *textual modeling tool* and *code generator*

- Adds modeling to Java,. C++, PHP: **Generates real systems**
- A sample of features
  - —Referential integrity on associations
  - —Code generation for patterns
  - —Blending of conventional code with models
  - —Infinitely nested state machines, with concurrency
  - —Separation of concerns for models: mixins, traits, mixsets, aspects

Platform and IDE agnostic:

- Command line compiler
- Web-based tool (UmpleOnline) for demos and education
- Plugins for Eclipse, MS Visual Studio Code, and other tools

Umple History

Developed *in itself*, 95% by students

- PhD (6), Masters (4), Capstone (50), Co-op/Intern (10)
- Has own parsing library (replacing Antlr), generator (replacing Jet)

- Initial version 2007 (IBM funding)
- Open source since 2011 (Google, Facebook funding)

- Online use / year (does not include downloaded use)
  - About 170K web visits
  - About 3 million compilations of systems

Key Websites

Entry-point: https://www.umple.org

Github: https://github.com/umple/umple

Umple Online: https://try.umple.org

Feature sets in Umple we will briefly look at

- Umple philosophy (next 2 slides)
- Tools: UmpleOnline and command line (slide 17 - 25)
- Modeling using class models that also incorporate classic code (slide 26 - 43) with teaching examples
  - —Attributes, Associations, Methods, Patterns, Constraints
- Modeling behaviour using state models (Slide 44 - 54) with teaching examples
  - —States, Events, Transitions, Guards, Nesting, Actions, Activities
  - —Concurrency
- Separation of Concerns in models (Slide 55 - 72) with teaching examples
  - —Mixins, Traits, Aspects, Mixsets

Umple Philosophy 1-4

P1. Modeling is programming and vice versa

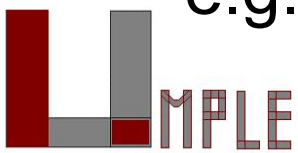
P2. An Umple programmer should never need to edit generated code to accomplish any task.

P3. The Umple compiler can accept and generate code that uses nothing but UML abstractions.

- The above is the inverse of the following

P4. A program without Umple features can be compiled by an Umple compiler.

- e.g. input Java results in the same as output



Umple Philosophy 5-8

P5. A programmer can incrementally add Umple features to an existing program

- Umplification

P6. Umple extends the base language in a minimally invasive and safe way.

P7. Umple features can be created and viewed diagrammatically or textually

P8. Umple goes beyond UML

I Invite you to follow along with examples: What technology will you need?

As a minimum: Any web browser.

For a richer command-line experience

- A computer (laptop) with Java 8-14 JDK
- Mac and Linux are the easiest platforms, but Windows also will work
- Download Umple Jar at http://dl.umple.org

You can also run Umple in Docker: http://docker.umple.org

The User Manual and Hello World

Go to http://helloworld.umple.org

Look at the first example

- Observe: just a plain method

view plain copy to clipboard print ?

```
01. /*
02.  * Simple Hello World example for Umple.
03.  * Compile this with Umple and it will generate Java
04.  * that is essentially the same.
05.  *
06.  * You could just as readily compile this code directly
07.  * with javac. However, this serves as the starting point:
08.  * Other examples in this manual show other things you
09.  * can do with Umple
10.  */
11. class HelloWorld {
12.   public static void main(String [ ] args) {
13.     System.out.println("Hello World");
14.   }
15. }
16.
17.
18.
```

Exercise: Compiling and Changing a Model

Look at the example at the bottom of
http://helloworld.umple.org (also on next slide)

- Observe: attribute, association, class hierarchy, mixin

Click on Load the above code into UmpleOnline

- Observe and modify the diagram
- Add an attribute
- Make a multiplicity error, then undo
- Generate code and take a look
- Download, compile and run (Follow as I show you how)

Demo of Compiling on the Command Line - 1

To compile on the command line you need Java 8 or higher

Download umple.jar from http://dl.umple.org

in Mac/Linux

- put it in a directory ~/tmp
- `alias umple='java -jar ~/tmp/umple.jar'`

Demo of Compiling on the Command Line - 2

Basic compilation

- `java -jar  umple.jar test.ump`
- `umple test.ump`

- `umple --help`

To generate and compile the java to a final system

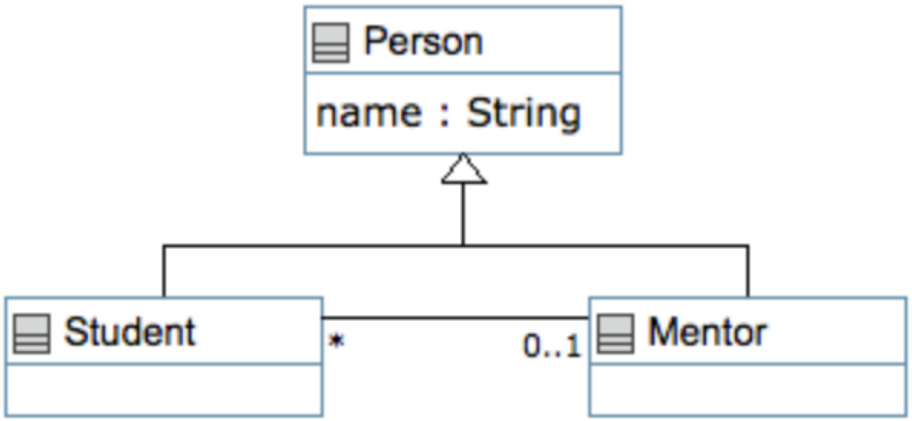
- `umple model.ump -c -`

To take any text from Umple online and compile it to Java on Mac or Linux **control-o** and then **paste into any terminal**

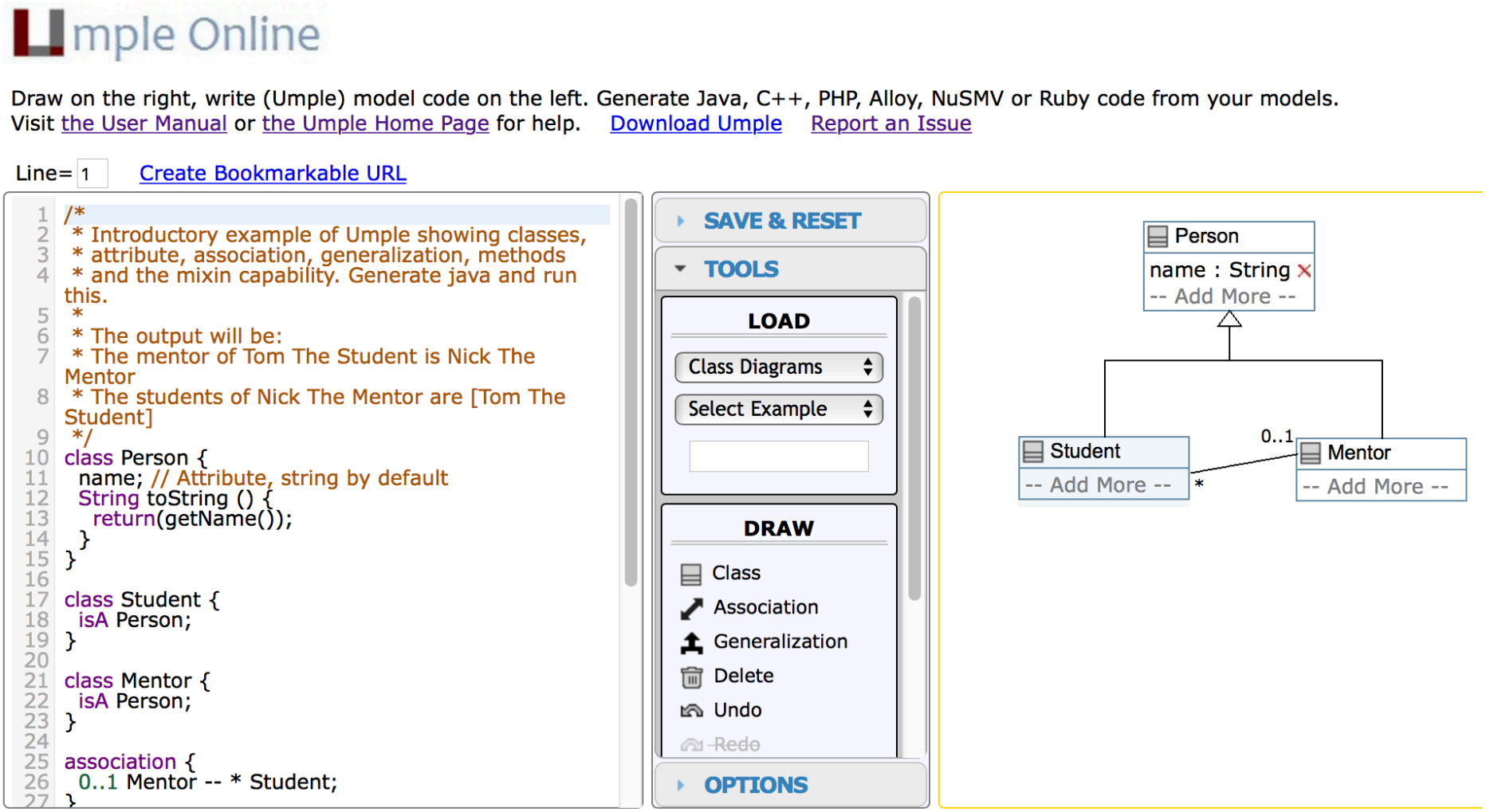
- Example on next slide

Hello World Example 2 in the User Manual

```
10.  class Person {
11.    name; // Attribute, string by default
12.    String toString () {
13.      return(getName());
14.    }
15.  }
16.
17.  class Student {
18.    isA Person;
19.  }
20.
21.  class Mentor {
22.    isA Person;
23.  }
24.
25.  association {
26.    0..1 Mentor -- * Student;
27.  }
28.
29.  class Person {
30.    // Notice that we are defining more contents for Person
31.    // This uses Umple's mixin capability
32.
33.    public static void main(String [ ] args) {
34.      Mentor m = new Mentor("Nick The Mentor");
35.      Student s = new Student("Tom The Student");
36.      s.setMentor(m);
37.      System.out.println("The mentor of "  + s  + " is " +  s.getMentor());
38.      System.out.println("The students of " +  m  + " are " +  m.getStudents());
39.    }
40.  }
```



Hello World Example 2 in UmpleOnline



Exploration of UmpleOnline

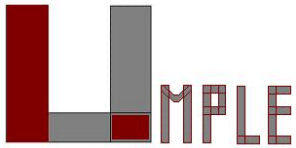
Explore class diagram examples

Explore options

- View – hide and showing text, diagram
  —Control-t (text), control-d (diagram) as shortcuts
- View – hide and showing methods, attributes
- Generate different default diagram types
  —Control-g (Graphviz), control-s (state), control-e

Generate code and look at the results

- In Umple you *never should modify generated code*, but it is designed to be readable for education and certification



Walkthrough of parts of the User Manual

https://manual.umple.org

Note in particular

- Key sections: attributes, associations, state machines
- Grammar
- Generated API
- Errors and warnings
- Editing pages in Github

Attributes

As in UML, **more abstract than fields or instance variables**

- Always private by default
- Accessed by get, set methods

- Can be *stereotyped* (we will skip in this tutorial) to affect code generation

- Can have *aspects* applied (discussed later)
- Can be *constrained* (discussed later)

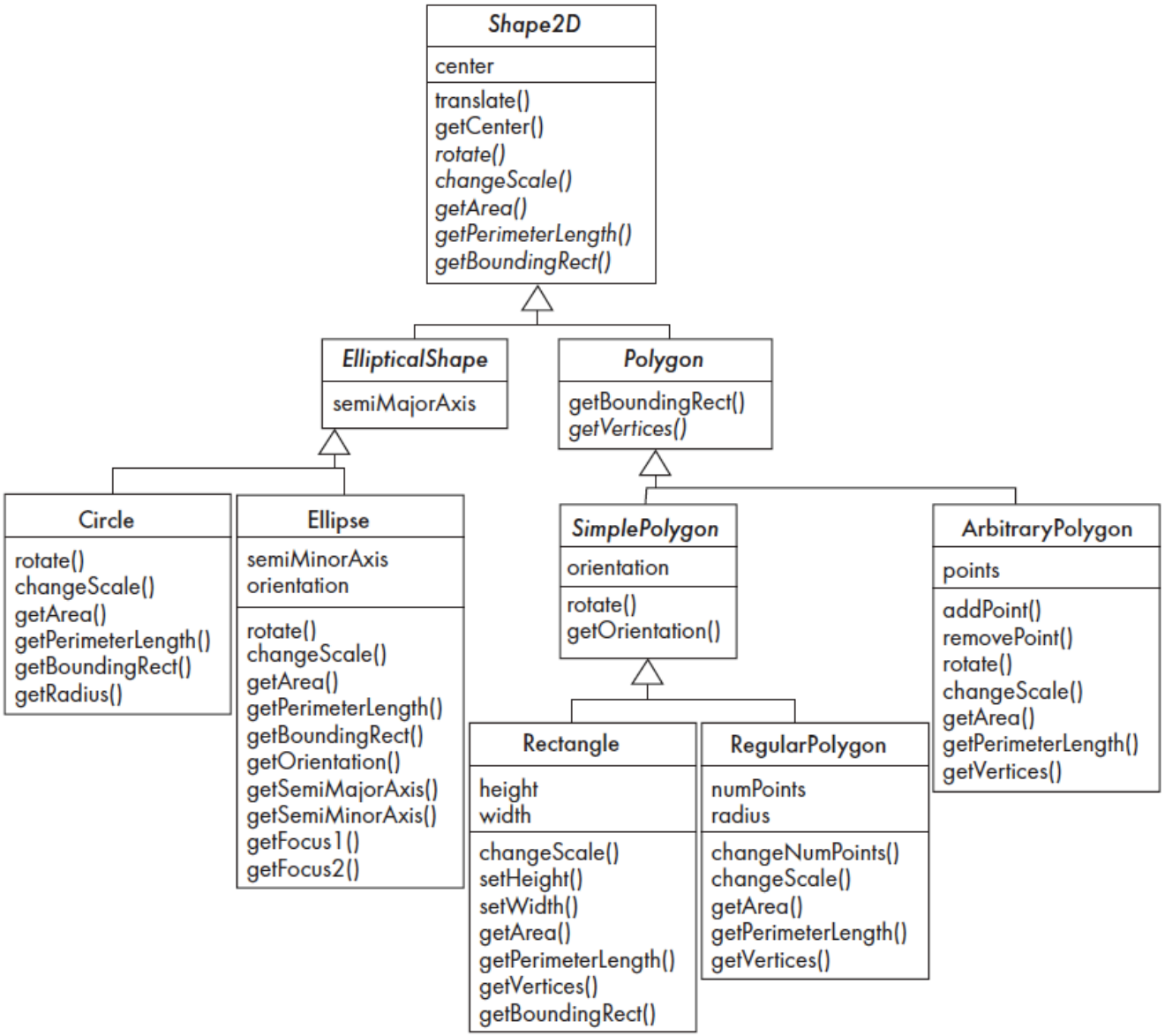
Generalization in Umple

Umple uses the isA keyword to indicate generalization

- Used to indicate superclass, used trait, implemented interface

```
class Shape {
  colour;
}
class Rectangle {
  isA Shape;
}
```

Example teaching material: Starting with the first Umpleonline Example, implement this



User-Written Methods in Umple

Methods can be added to any Umple code.

Umple *parses the signature only; the rest is passed to the generated code*.

You can specify different bodies in different languages

We will look at examples in the user manual …

Associations in Umple

```
class Employee {
  id;
  firstName;
  lastName;
}

class Company {
  name;
  1 -- * Employee;
}
```

Referential Integrity

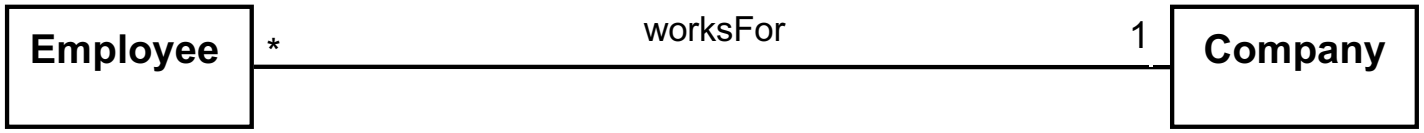
When an instance on one side of the association changes

- The linked instances on the other side know …
- And vice-versa

This is standard in Umple associations, which are bidirectional

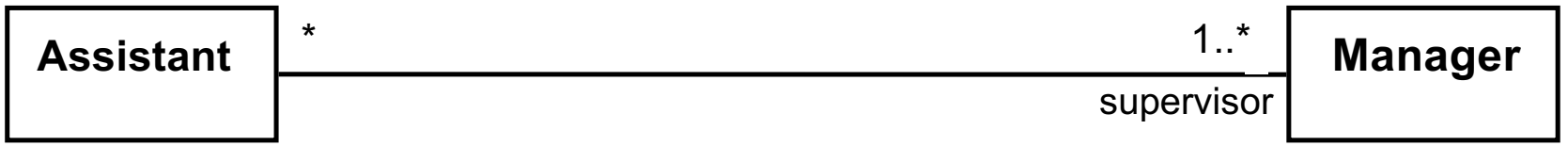
Example Teaching Material: Analyzing and validating associations

- **Many-to-one**
  - —A company has many employees,
  - —An employee can only work for one company.
    - This company will not store data about the moonlighting activities of employees!
  - —A company can have zero employees
    - E.g. a ‘shell’ company
  - —It is not possible to be an employee unless you work for a company



Example Teaching Material: Analyzing and validating associations

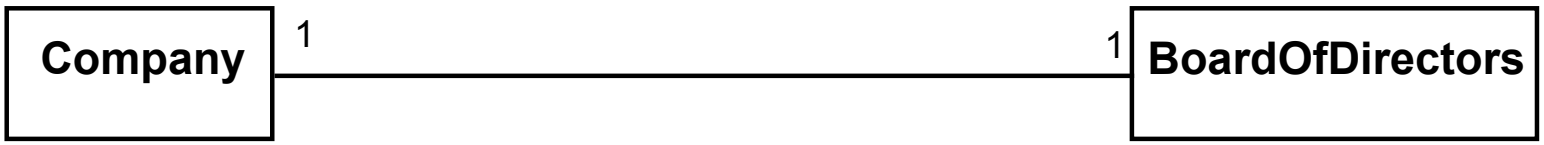
- **Many-to-many**
  - —An assistant can work for many managers
  - —A manager can have many assistants
  - —Assistants can work in pools
  - —Managers can have a group of assistants
  - —Some managers might have zero assistants.
  - —Is it possible for an assistant to have, perhaps temporarily, zero managers?



Open in Umple

Example Teaching Material: Analyzing and validating associations

- **One-to-one**
  - —For each company, there is exactly one board of directors
  - —A board is the board of only one company
  - —A company must always have a board
  - —A board must always be of some company



Open in Umple

Unidirectional Associations

Associations are by default *bi-directional*

Limit the navigability direction of an association by adding an arrow at one end

In the following unidirectional association

— A Day knows about its notes, but a Note does not know which Day is belongs to

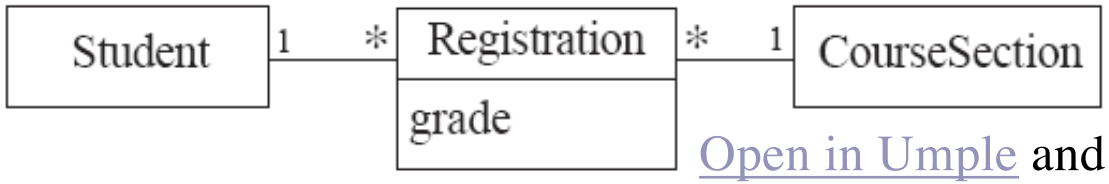
—Note remains ‘uncoupled’ and can be used in other contexts

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

Open in Umple

Association Classes

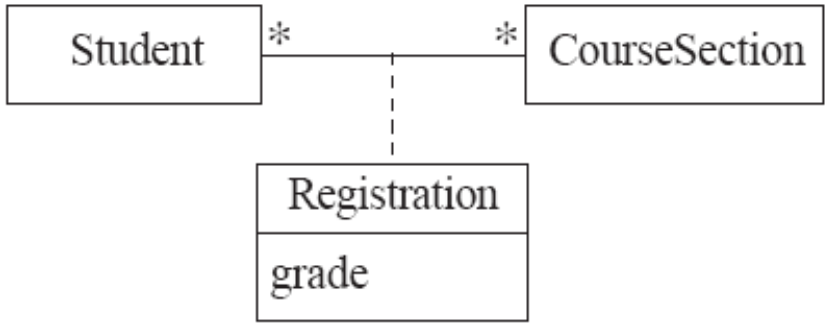
Sometimes, an attribute that concerns two associated classes cannot be placed in either of the classes



Open in Umple and extended example

The following are nearly equivalent

- The only difference:
  - —in the association class there can be only a *single* registration of a given Student in a CourseSection



Association Classes (cont.)

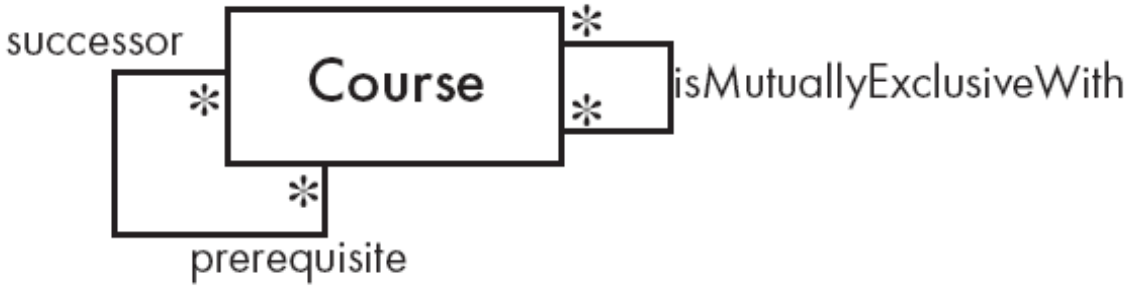
Umple code

```
class Student {}
class CourseSection {}
associationClass Registration {
  *  Student;
  *  CourseSection;
}
```

Open in UmpleOnline, and then generate code

Reflexive Associations

An association that connects a class to itself




```
class Course {
   * self isMutuallyExclusiveWith; // Symmetric
}

association {
     * Course successor -- * Course prerequisite;
}
```

Open in Umple

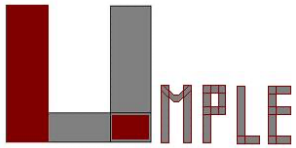
Singleton Pattern

Standard pattern to enable only a single instance of a class to be created.

- private constructor
- getInstance() method

Declaring in Umple

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



Example Teaching Material: Assignment

This assignment will enable you to learn about and gain experience with:

- Basic of user stories for a case study
- The basics of class diagrams
- Umple as a modeling tool

Building a simple system from a class diagram

As you are working on this assignment, if you need help you can refer to the course notes and videos for Chapters 2 (object orientation) and Chapter 5 (Class diagrams). You should also refer to the Umple user manual http://manual.umple.org

If you want to ask questions, you should ask them on the Microsoft Teams channel for Assignment 2

The Case study you will be working with is Case study 2: Test and contact tracing for a pandemic. In Chapter 4 we looked at that and listed some of the actors (types of users). You will need to review our discussion in the class. Feel free also to do some domain analysis by searching the web to learn a little more about this domain.

Example Teaching Material: Assignment

The following are the core requirements you will be working with in this assignment (this is for the first version of the system): You will build a system to

Record information about tests for the virus and tests for antibodies.

Each test has at least a date, time, location, test type (i.e. manufacturer, type and version of test), a test location (test center, hospital etc), a person tested (patient), a lab processing the test, a date the test was processed, and an outcome.

Each patient has a name, address, phone numbers and health number.

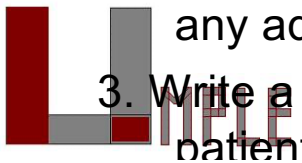
Each patient has a list of contacts (other people), and a set of dates when the patient was in contact with the other people.

Each patient also has a list of symptoms, where each symptom has a start and end date.

Feel free to add more details, state assumptions and clarify these requirements.

Instructions

1. Write 5 user stories for the core requirements listed above.
2. Using Umple, create a class diagram to model the requirements listed above (including any adjustments to the requirements you decide to make as part of your research).
3. Write a main program embedded in the Umple that will: a) Create an instance of a patient by calling the Umple-generated constructor for class Patient, b) Create two



Example Teaching Material: Assignment

Feel free to add more details, state assumptions and clarify these requirements.

Instructions

1. Write 5 **user stories** for the core requirements listed above.
2. **Using Umple, create a class diagram to model the requirements** listed above (including any adjustments to the requirements you decide to make as part of your research).
3. **Write a main program embedded in the Umple** that will
   a) Create an instance of a patient by calling the Umple-generated constructor for class Patient,
   b) Create two instances of a tests (on different dates) administered to that patient that are properly linked to the patient using the Umple-generated API;
   c) Print out neatly the patient information by calling a method on the instance of the Patient class; this method will automatically also print out the test information neatly for the tests the patient has had.

Submit your user stories, your Umple code, an image of your Umple class diagram, and a transcript of you running your code.

As with other work in this course, there won't be hard deadlines (except the last day of the course), but if you get the work done before June 15 there will be a bonus, and the value of the bonus will diminish in the days following June 15th.

Basic Constraints

Shown in square brackets

- Code is added to the constructor and the set method

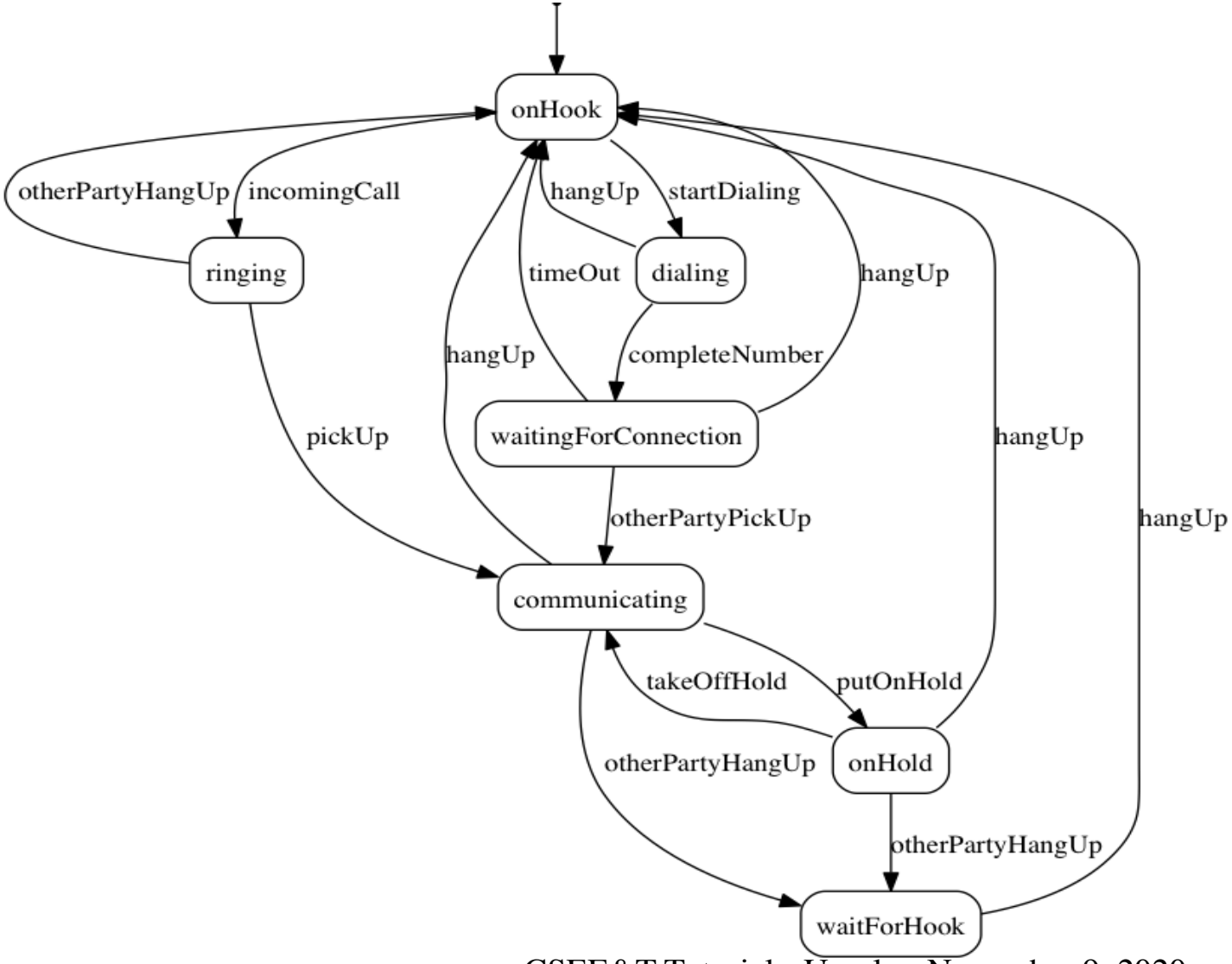
```
class X {
  Integer i;
  [! (i == 10)]
}
```

We will see constraints in state machines (as guards)

Basics of State Machines

- At any given point in time, the system is in one state.
- It will remain in this state until an event occurs that causes it to change state.
- Standard UML notation and semantics

State Machine with tracing:
Phone and Lines example in UmpleOnline



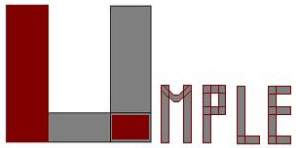
Do Activities and Concurrency

A do activity executes

- In a separate thread
- Until
  - —Its method terminates, or
  - —The state needs to exit (killing the tread)

Example uses:

- Outputting a stream (e.g. playing music)
- Monitoring something
- Running a motor while in the state
- Achieving concurrency, using multiple do activities



State Tables and Simulations

Allow analysis of state machines statically without having to write code

We will explore these in UmpleOnline by looking at state machine examples and generating tables and simulations

Example Teaching Material: State Machine / Umple Tasks Case Study: Dishwasher

We will start with the Dishwasher example in UmpleOnline

- I will set up a task
- Load the task and change it to increase the regular wash time by 2 minutes (simulated by 2 seconds)
- Compile on the command line to check it works
- Submit the task

We will use UmpleOnline's Task capability to ask you to make a change to the dishwasher example

Example teaching material: Assignment

In this assignment 4 you will create simulation of an oven. You will primarily use state machines for this, along with some additional code to run the simulation.

The page to submit the assignment can be found by selecting 'Assignment 4' from the Assignments tab at the top of the Brightspace page for the course.

The oven will be a combination Microwave/Convection oven with **the following features**.

It will have a radiant heating element that can radiate over the food to brown it, and also generate hot air for cooking. This has two power levels (low and high, where low is half the power of high). Default is high.

It will have a fan that turns on to circulate hot air. The fan should always be on during any cooking operation.

It will have a microwave emitter that emits microwaves for heating. This can be set at two power levels (low and high, where low is half the power of high), default high. Cooking at other powers can be accomplished by turning the emitter on and off repeatedly (e.g. 1/6 power can be achieved by turning it on at low power for 2 seconds, and the off for 4 seconds, then back on for 2 seconds and so on).

It has a beeper that can make a sound for a second.

It has an alphanumeric display with up to 10 characters to display cooking times and other messages..

It has a door sensor (for safety, microwave heating must be turned off if the door is open, and radiant heating must be turned off if the door is open for more than 3 minutes).

It has an air temperature sensor that is used to control the temperature (e.g. to keep it at 200 degrees C) during cooking, and also to ensure that all heating is turned off if the temperature exceeds 400 degrees C.

It has an exhaust vent that opens to let air flow out during microwave-only cooking but is otherwise kept closed.

Example teaching material: Assignment

User stories for the oven include the following. During UI design, you may modify some of these.

1. Setting a power level, selecting a time, and pressing start to use the microwave function at full power. The order of whether the time is set first, or the power level is set first does not matter, but the oven must be in idle state first. Three beeps will be sounded when cooking ends.

2. Suspending microwaving by opening the door or pressing a button.

3. Cancelling all cooking by pressing a button (or maybe pressing a button twice)

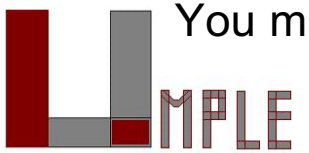
4. Setting a temperature, selecting a time, and pressing start to use the convection feature.

5. Suspending convection cooking by pressing a button.

6. Cooking with both microwave and convection at the same time.

7. A timer function that allows timing without any cooking. 4 Beeps will be sounded when the timer runs out.

You may add other user stories.



Example teaching material: Assignment

Deliverables: Your task in this assignment is as follows:

1. Design a ***user interface*** consisting of buttons that can control the various features of the oven, and the messages that will be output to the display. Update the user stories as needed. Draw an image of what your user interface will look like. Hand this in as a Word or pdf file. You will not program the user interface.
2. Create a ***state machine model*** in Umple that will model the control of the user interface and the cooking process.

Hints:

a) Each button is an event (i.e. specified in a state machine, and where a method call for the event is generated when the state machine is compiled)

b) Opening and closing the door are also an events.

c) Some transitions are only allowed in certain situations (e.g. you can't start cooking if the door is open).

d) Your state machine should call a display() method to display output to the user (see item 3 c i below).

e) Your state machine should also call methods that would start and stop the cooking (microwave and convection) and the fan, as well as open or close the exhaust vent and sound a beep. These may be events triggering transitions in one or more different state machines.

f) Use the 'after(n)' event to enable timing. There is sample code in the user manual and below to explain how to set up time delays in Umple state machines. We discussed this in class too.

g) You may need separate state machines to control the user interface and the cooking. You can prefix a state machine with the 'queued' keyword to avoid the state machines blocking each other.

Example teaching material: Assignment

Create a ***simulation main program*** in Umple whereby the user can enter the various events on the console.

a) The user should be able to type letters to simulate pressing buttons, as well as opening and closing the door.

b) The user should also be able to type commands that would simulate reaching a certain temperature.

c) The output of the simulation should include two types of textual output:

- i. What appears on the display, prefixed by 'DISPLAY: '
- ii. other actions occurring (heating on and off; beeping, vent opening or closing) at specific times, such as 3:30:02: Radiant heat on high.

Model-Based Template Generation of Text

Allow output of complex text in any class

- Can generate XML, html, code, UI, etc.

Template for exactly the content

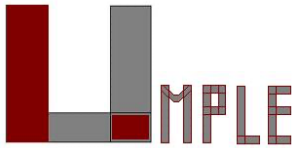
- `textToOutout` **`<<!output this!>>`**

Expression

- **`<<=someCode();>>`**

Internal logic within a template

- **`<<# if(a==0){#>>`** … **`<<#}#>>`**



Template Generation - Continued

We will look at examples in the in the User Manual

- Simple multiplication table
- Form letter

Mixins: Motivation

Product variants have long been important for

—Product lines/families, whose members target different:

- hardware, OS, feature sets, basic/pro versions

—Feature-oriented development (separation of concerns

Separation of Concerns by Mixins in Umple

Mixins allow incremental addition to a class of attributes, associations, state machines, and any other feature

Example:

```
class X { a; }
class X { b; }
```

- The result would be a class with both a and b.

It doesn't matter whether the mixins are

- Both in the same file
- One in *one file*, that includes the other in another file
- In two separate files, with a third file invoking them

Advantages of Mixins

Smaller files that are easier to understand

Possibility to define *variants* and *product lines*:

- Different versions of a class for different software versions (e.g. a professional version)

Aspects: Motivation

We often don't quite like the code as generated

Or

We want to do a little more than what the generated code does

Or

We want to inject some feature (e.g. security checks) into many places of generated or custom code

Aspects: General concepts

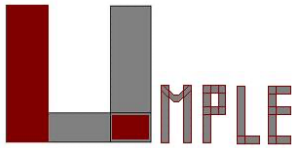
Create a *pointcut* that specifies (advises) where to inject code at multiple points elsewhere in a system

- The pointcut uses a *pattern to match where to inject*
- Pieces of code that would otherwise be scattered are thus gathered into the aspect

But: There is potentially acute sensitivity to change

- If the code changes the aspect may need to change
- Yet without tool support, developers wouldn't know this

Drawback: Delocalization even stronger than for mixins



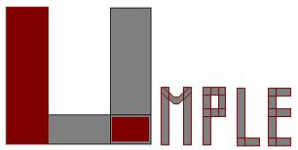
Aspect Orientation in Umple

It is common to limit a pointcuts a single class

- Inject code before, after, or around execution of custom or generated methods and constructors

```
class Person {
  name;
  before setName {
    if (aName != null && aName.length() > 20) { return false;
    }
  }
}
```

We have found these limited aspects nonetheless solve key problems



Traits: Motivation

We may want to inject similar elements into unrelated classes without complex multiple inheritance

- Elements can be
  - —Methods
  - —Attributes
  - —Associations
  - —States or state machines
  - —.. Anything

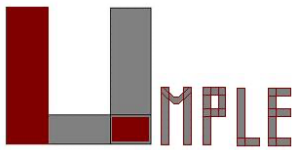
Separation of Concerns by Traits

Allow modeling elements to be made available in multiple classes

```
trait Identifiable {
  firstName;
  lastName;
  address;
  phoneNumber;
  fullName = {firstName + " " + lastName}
  Boolean isLongName() {return lastName.length() > 1;}
}

class Person {
  isA Identifiable;
}
```

See more complete version of this in the user manual



Associations in Traits: Observer Pattern

```
class Dashboard{
  void update (Sensor sensor){ /*implementation*/ }
}
class Sensor{
  isA Subject< Observer = Dashboard >;
}
trait Subject <Observer>{
  0..1 -> * Observer;
  void notifyObservers() { /*implementation*/ }
}
```

Mixins and Traits together

- Examples of mixins and traits combined in the user manual:

  —Mixins with traits:

  - https://cruise.umple.org/umple/TraitsandUmpleMixins.html

Mixsets: Motivations

A feature or variant needs to inject or alter code in many places

- Historically tools like the C Preprocessor were used
- Now tools like “Pure: Variants”

There is also a need to

- Enable *model* variants in a very straightforward way
- Blend variants with code/models in core compilers
  - —With harmonious syntax + analysable semantics
  - —Without the need for tools external to the compiler

Mixsets: Top-Level Syntax

Mixsets are named sets of mixins

```
mixset Name {
  // Anything valid in Umple at top level
}
```

The following syntactic sugar works for top level elements (class, trait, interface, association, etc.)

```
mixset Name class Classname {
}
```

Use Statements

A use statement specifies inclusion of either

- A file, or
- A mixset

```
use Name;
```

A mixset is conceptually a *virtual file* that is composed of a set of model/code elements

The use statement for a mixset can appear

- *Before*, *after* or *among* the definition of the mixset parts
- In *another mixset*
- On the *command line* to generate a variant

Mixsets and Mixins: Synergies

- The blocks defined by a mixset are mixins
- Mixsets themselves can be composed using mixins
  —e.g.

```
mixset Name1  {class X { a; } }
```

And somewhere else

```
mixset Name1  {class X { b; } }
use Name1;
```

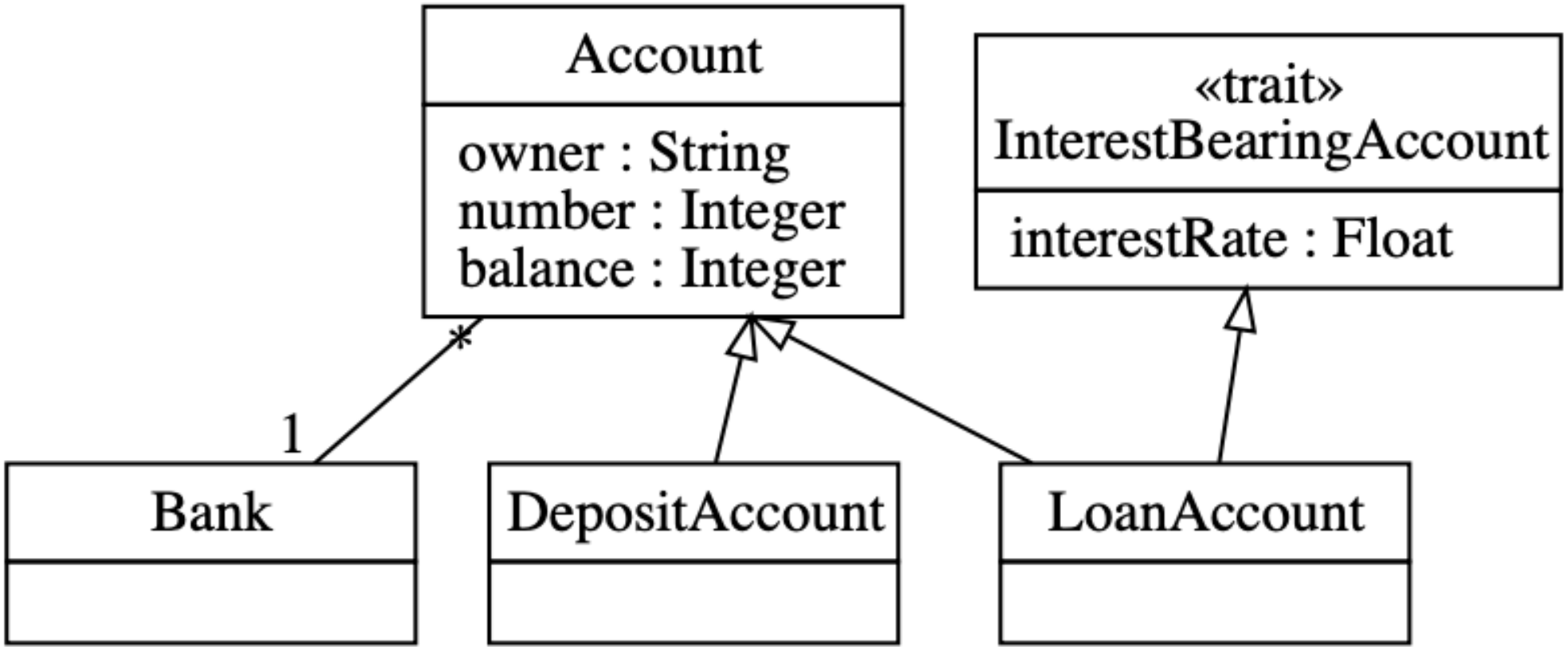
Would be the same as:

```
class X { a; b;}
```

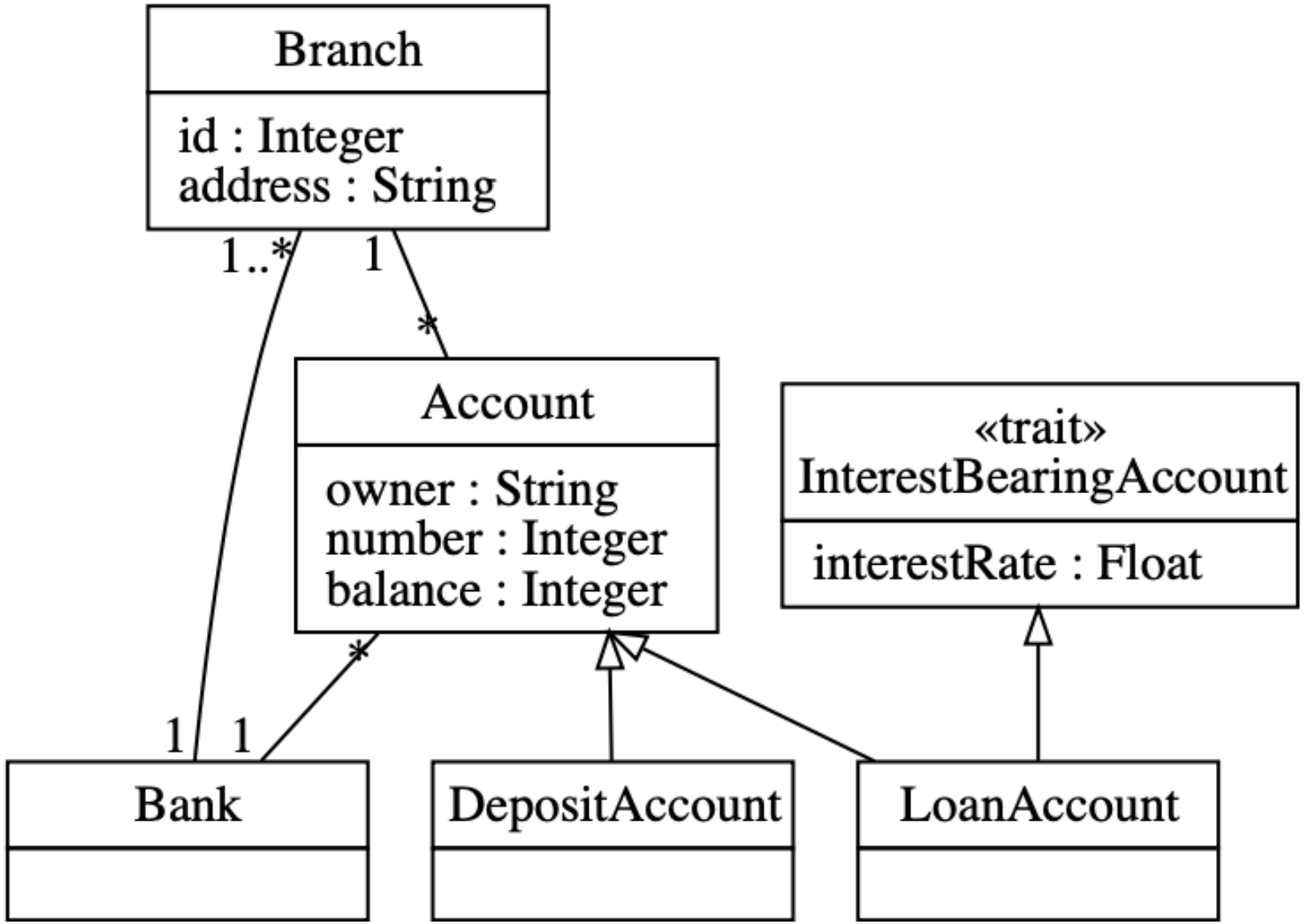
Example Teaching Material:
Umple Model/Code for Basic Bank

```
1   class Bank {
2     1 -- * Account;
3   }
4
5   class Account {
6     owner; Integer number; Integer balance;
7   }
8
9   trait InterestBearingAccount {
10    Float interestRate;
11  }
12
13  class DepositAccount {
14    isA Account;
15  }
16
17  class LoanAccount {
18    isA Account, InterestBearingAccount;
19  }
```

Class Diagram of Basic Bank Example: Generated from Umple

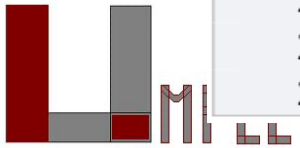


Adding Optional Multi-branch Feature



Example: Multi-branch Umple Model/Code

```
1   class Bank {
2     1 -- * Account;
3     mixset Multibranch 1 -- 1..* Branch;
4   }
5
6   mixset Multibranch class Branch {
7     Integer id; String address;
8   }
9
10  class Account {
11    owner; Integer number; Integer balance;
12    mixset Multibranch * -- 1 Branch;
13  }
14
15  trait InterestBearingAccount {
16    Float interestRate;
17  }
18
19  class DepositAccount {
20    isA Account;
21    mixset OverdraftsAllowed {
22      Integer overdraftLimit;
23      isA InterestBearingAccount;
24    }
25  }
26
27  class LoanAccount {
28    isA Account, InterestBearingAccount;
29  }
```



Part 3: Summary of teaching an undergrad course in Umple

Topics:

- Object oriented principles
- Class modeling
  —Attributes, Associations, Patterns
- Agility: Add small features, use of git with models
- Design-test-refactor cycle
- Constraints
- State machine modeling
- Concurrency
- Testing

Part 4: Using Umple as a case in a capstone course

I serve as the ‘customer’

Groups of 1-4 students are tasked to work on issues

- I point them to key documentation (next slide)
- I start them with a small bug (often experienced by end-users) or trivial enhancement
- I teach the whole class about agile approaches
- Students discuss progress every week with the class
- They submit pull requests that are reviewed by me and others
- They must do test-driven and model-driven development so PRs must have relevant updates to the model and tests

Material I ask Capstone Students and other contributors to look at

We will look at:

- Code in Github: http://code.umple.org
- Wiki and key pages: http://wiki.umple.org
- Architecture: http://architecture.umple.org
- Generated diagrams: http://metamodel.umple.org
- Generated Javadoc: http://javadoc.umple.org
- Sample master code
- Sample test output: http://qa.umple.org
- Sample code for generators (that replaced Jet)
- UmpleParser (that replaced Antlr

Some learning objectives for the capstone

Ability to understand and make successful changes a complex code base

Ability to perform model-driven development

Ability to perform test-driven development

Understand compiler design

Understand code generation

Understand parsing and grammars

Understand setup and running of a modern agile project

Many other things: Docker, devops, front end (javascript, UI design, etc).

Marking scheme for the capstone

20% Customer satisfaction

20% Project management / following agile process and professionalism (e.g. resolving issues when there are disputes with teammates)

20% Design (at the model and code level)

10% Presentations

20% Communication (including code commenting, updating wikis, recording design decisions, comments on issues)

10% Complexity (sufficient work done)

Teams can request that certain team members get different grades if all have not contributed equally

Conclusion
(Referring back to the introduction)

We need to teach agility better

- Umple blends with all agile processes

We need to teach design abstraction better

- Umple allows this to be done textually and graphically with a tool students find easy to use

Students need feedback on working systems

- Umple allows creation of working systems with real code
- It allows analysis of models
- Feedback comes back from the model compiler, embedded code compiler, and at runtime