COMP 249: Object Oriented Programming II

Unified Modeling Language (UML)

Introduction to UML

- UML is a software design tool that can be used within the context of any OOP language

- UML is a graphical language used for designing and documenting OOP software

UML

- Pseudocode is a way of representing a program in a linear and algebraic manner.
  - It simplifies design by eliminating the details of the programming language syntax.

- Graphical representation systems for program design have also been used.
  - *Flowcharts* and *structure diagrams* for example.

- *Unified Modeling Language* (*UML*) is yet another graphical representation formalism.
  - UML is designed to reflect and be used with the OOP philosophy.

History of UML

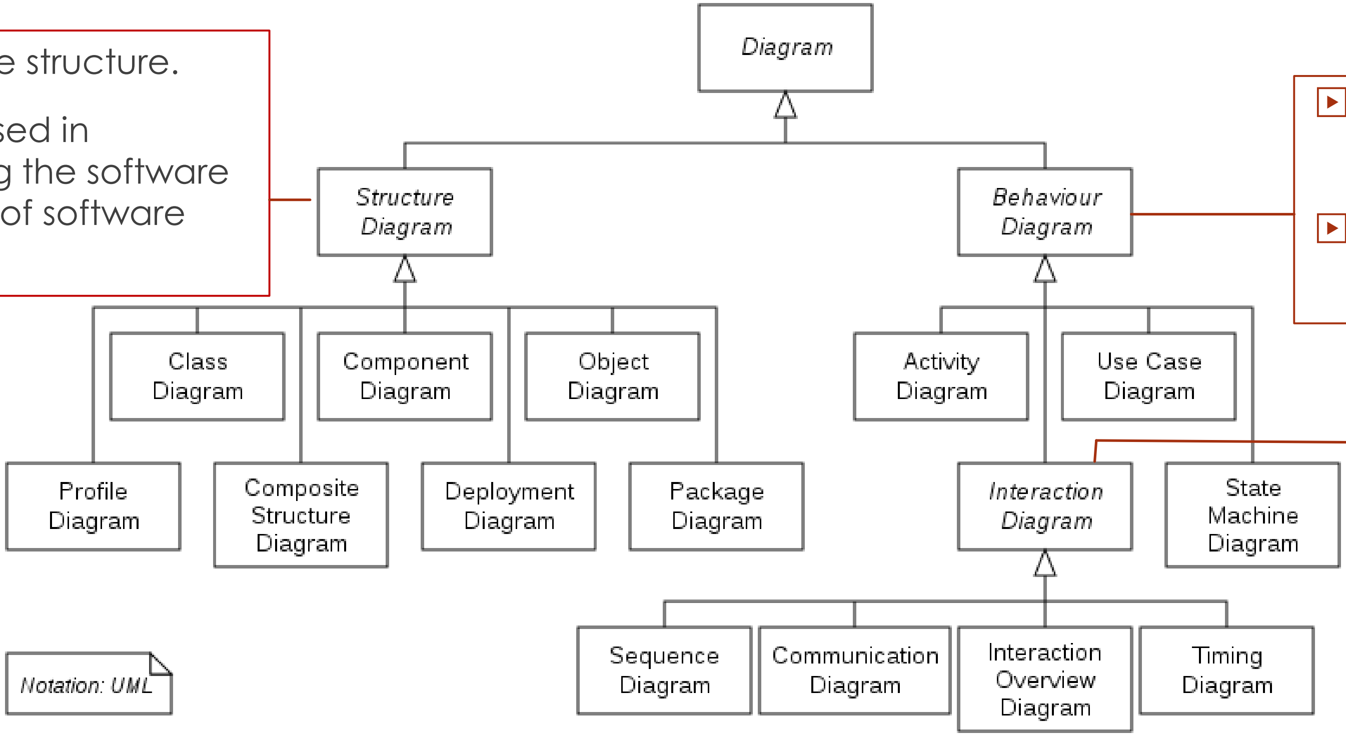
- As OOP has developed, different groups have developed graphical or other representations for OOP design.

- In 1996, Brady Booch, Ivar Jacobson, and James Rumbaugh released an early version of UML.
  - Its purpose was to produce a standardized graphical representation language for object-oriented design and documentation.

- Since then, UML has been developed and revised in response to feedback from the OOP community.
  - Today, the UML standard is maintained and certified by the Object Management Group (OMG).

UML Diagrams

- Represent the structure.
- Extensively used in documenting the software architecture of software systems.



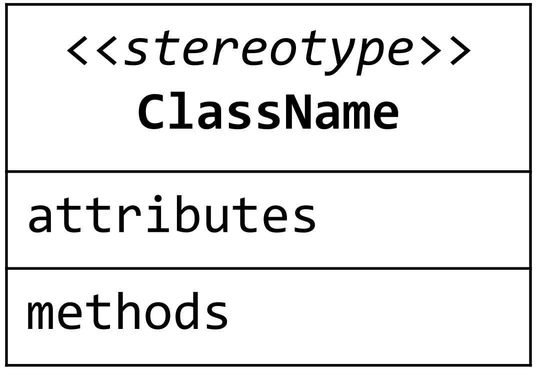
- Emphasize what must happen in the system being modeled.
- Extensively used to describe the functionality of software systems.

- Emphasize the flow of control and data among the things in the system being modeled.
- May show how objects communicate with each other (i.e. sequence diagram.

UML Class Diagrams (1)

- Classes are central to OOP, and the *class diagram* is the easiest of the UML graphical representations to understand and use

- A class diagram is divided up into three sections
  - The top section contains the class name and applicable stereotypes (*<<abstract>>*, *<<interface>>*…)
  - The middle section contains its data members
  - The bottom section contains its methods



<<stereotype>>

ClassName

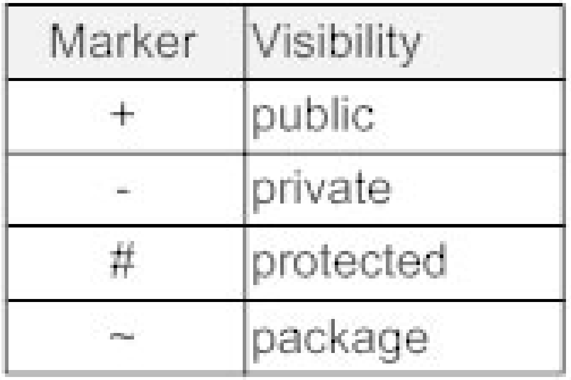
attributes

methods

UML Class Diagrams (2)

- The data specification for each piece of data in a UML diagram consists of its name, followed by a colon, followed by its type

- Each name is preceded by a character that specifies its access type (visibility):
  - A minus sign (-) indicates private access
  - A plus sign (+) indicates public access
  - A sharp (#) indicates protected access
  - A tilde (~) indicates package access



Marker	Visibility
+	public
-	private
#	protected
~	package

UML Class Diagrams (3)

- Each method in a UML diagram is indicated by the name of the method, followed by its parenthesized parameter list, a colon, and its return type

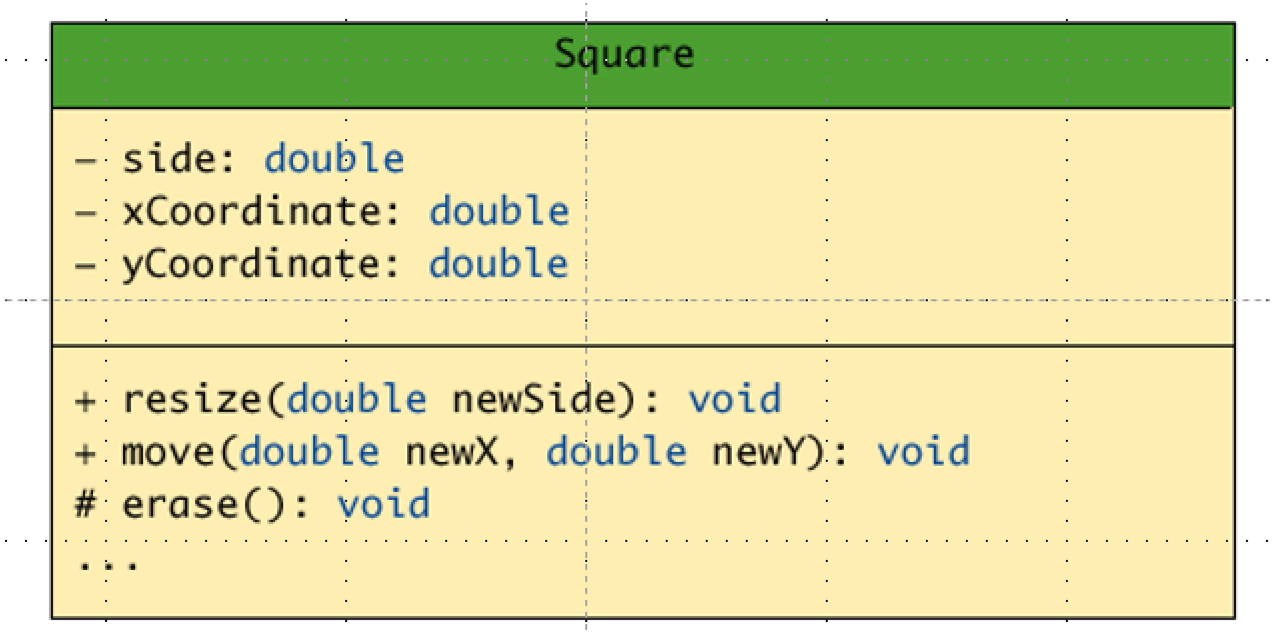
- The access type of each method is indicated in the same way as for data

UML Class Diagrams (4)

- A class diagram do not need to give a complete description of the class
  - If a given analysis does not require that all the class members be represented, then those members are not listed in the class diagram
  - Missing members (those irrelevant to the current description) are indicated with an ellipsis (three dots)

UML Class Diagrams (5)

An example of a UML class Diagram:



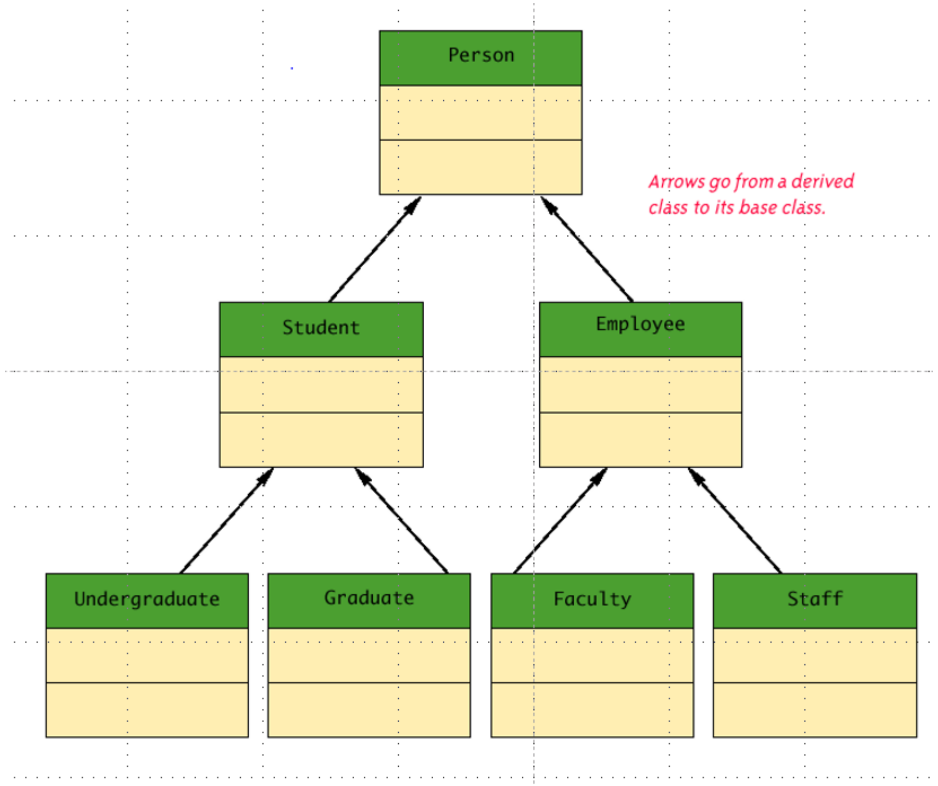
```
Square

– side: double
– xCoordinate: double
– yCoordinate: double

+ resize(double newSide): void
+ move(double newX, double newY): void
# erase(): void
...
```

Class Interactions (1)

- Rather than show just the interface of a class, class diagrams are primarily designed to show the interactions among classes
- UML has various ways to indicate the information flow from one class object to another using different sorts of annotated arrows
- UML has annotations for class groupings into packages, for inheritance, and for other interactions
- In addition to these established annotations, UML is extensible

Class Interactions (2)

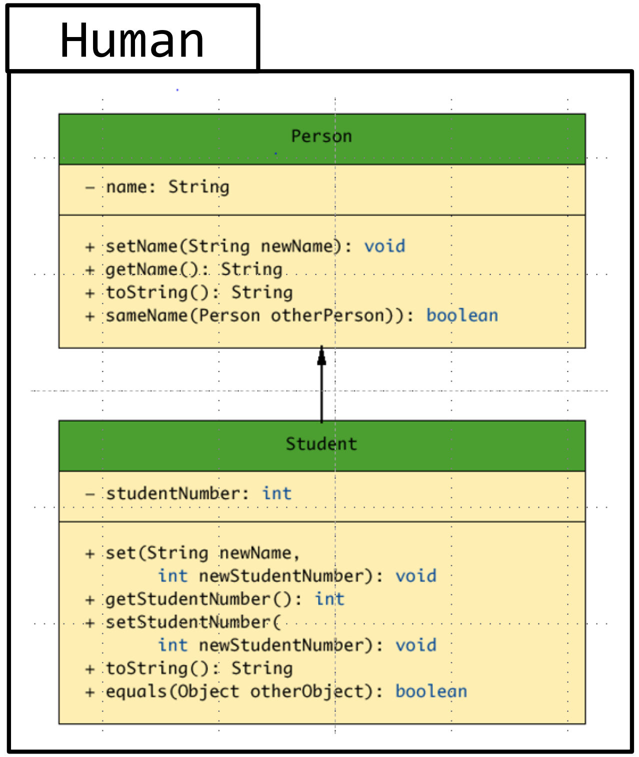
- To represent inheritance between classes:
  - Each base class is drawn above its derived class(es)
  - An upward pointing arrow is drawn between them to indicate the inheritance relationship. The arrows also help in locating method definitions.



Arrows go from a derived class to its base class.

Class Interactions (3)

- Packages can be represented in a class diagram by a rectangle with the package name
- All the member classes are placed within the rectangle



```
Human

Person
- name: String
+ setName(String newName): void
+ getName(): String
+ toString(): String
+ sameName(Person otherPerson)): boolean

Student
- studentNumber: int
+ set(String newName,
      int newStudentNumber): void
+ getStudentNumber(): int
+ setStudentNumber(
      int newStudentNumber): void
+ toString(): String
+ equals(Object otherObject): boolean
```

Software tools

Many software tools can be used to draw UML diagrams such as:

- Microsoft Visio
- Smart Draw
- ObjectAid plugin for Eclipse

Some tools are available online:

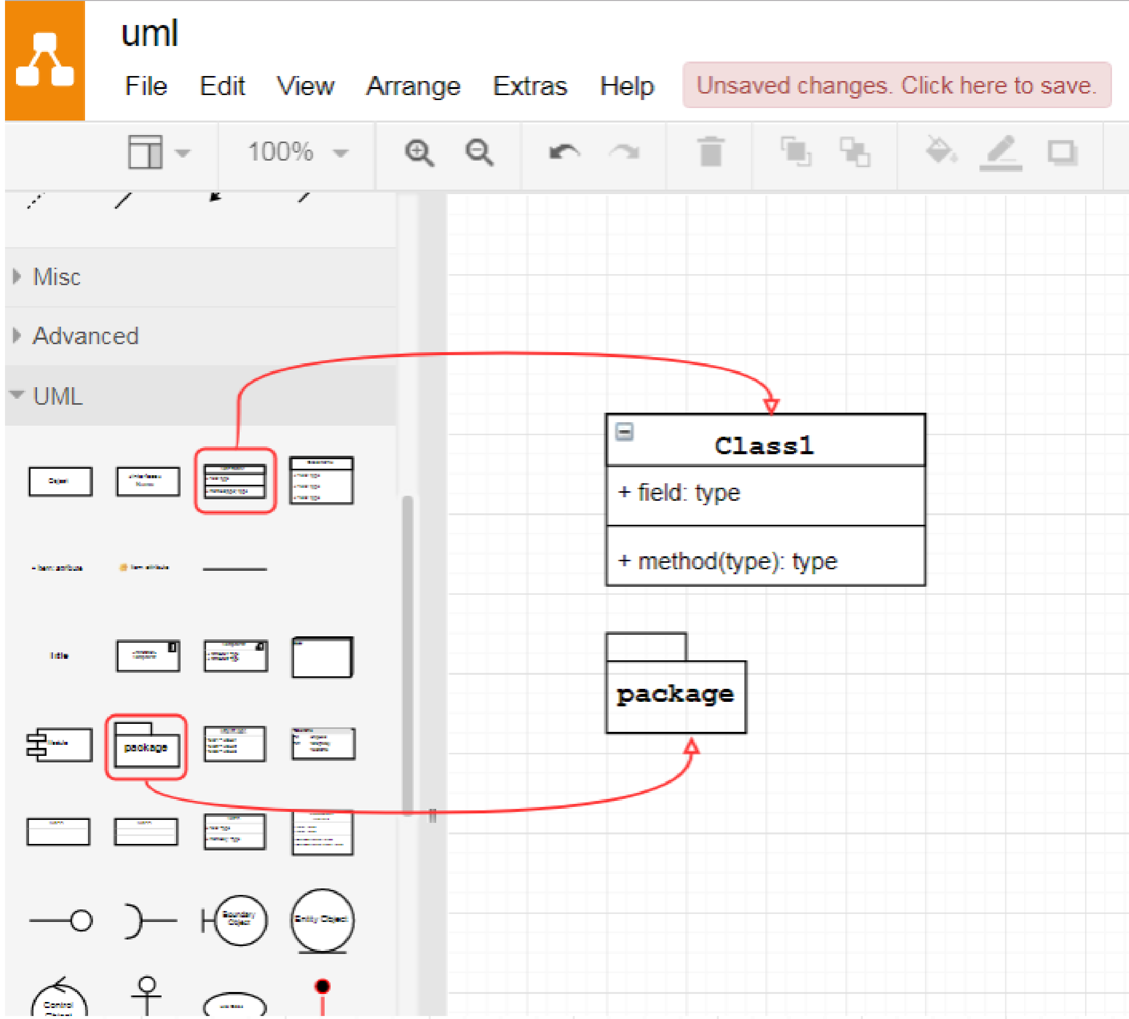
- draw.io (`https://www.draw.io/`)

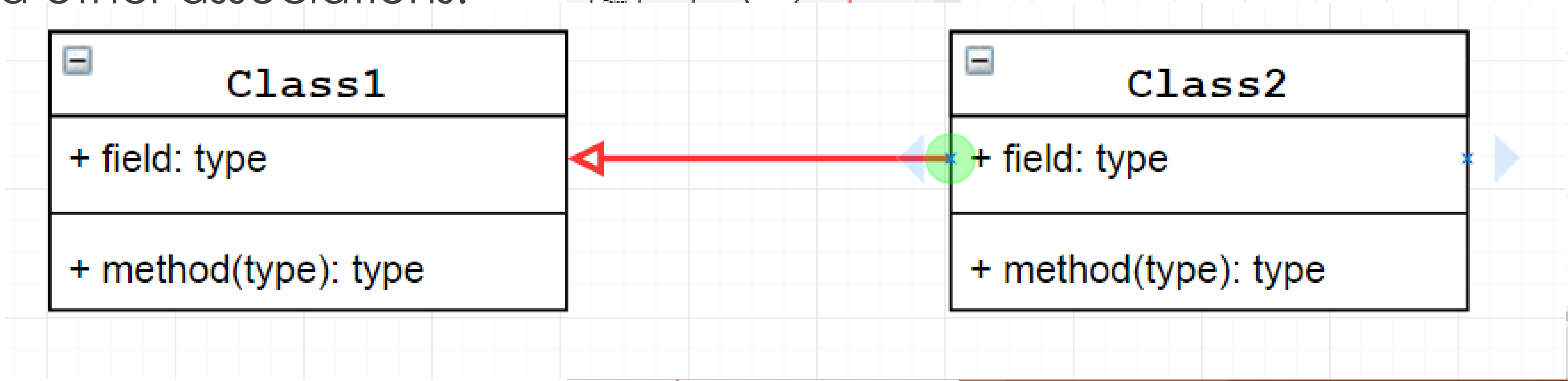
Using draw.io

Use the UML drop-down menu.

Most of the common shapes are already defined.

By hovering over a shape and dragging a corner over to another shape, you can create connectors, which can then be used to represent inheritance and other associations.


uml
File
Edit
View
Arrange
Extras
Help
Unsaved changes. Click here to save.
100%
Misc
Advanced
UML
package
Class1
+ field: type
+ method(type): type
package


Class1
+ field: type
+ method(type): type
Class2
+ field: type
+ method(type): type