COMP249 Tutorial 3: UML Class Diagrams and OOP Concepts, Continued

Ella Noyes

July 16, 2024

1 Elaborating on Abstract Classes and Interfaces

I added some simple code to the GitHub repository to demonstrate how to work with an abstract class and an interface. We'll look through that together.

2 Class Composition

Class composition occurs when one class contains an instance of another class as one of its members. This is sometimes referred to as a has-a relationship. For example, you might want to create a Library class that has an associated Librarian and a list of Books.

```
public class Book {
    private String title;
    private String author;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public void displayInfo() {
        System.out.println("Title: " + title + ", Author: " + author);
    }
}


public class Librarian {
    private String name;

    public Librarian(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
```

```
    public void assistPatron() {
        System.out.println(name + " is assisting a patron.");
    }
}


public class Library {
    private String name;
    private List<Book> books;
    private Librarian librarian;

    public Library(String name, Librarian librarian) {
        this.name = name;
        this.books = new ArrayList<>();
        this.librarian = librarian;
    }

    public void addBook(Book book) {
        books.add(book);
    }

    public void displayCatalogue() {
        for (Book book : books) {
            book.displayInfo();
        }
    }

    public void assistPatron() {
        librarian.assistPatron();
    }
}
```

The Library class "has-a" Librarian, and "has-a" (or has many!) Book.

3 UML Class Diagram Basics

3.1 Class Compartments

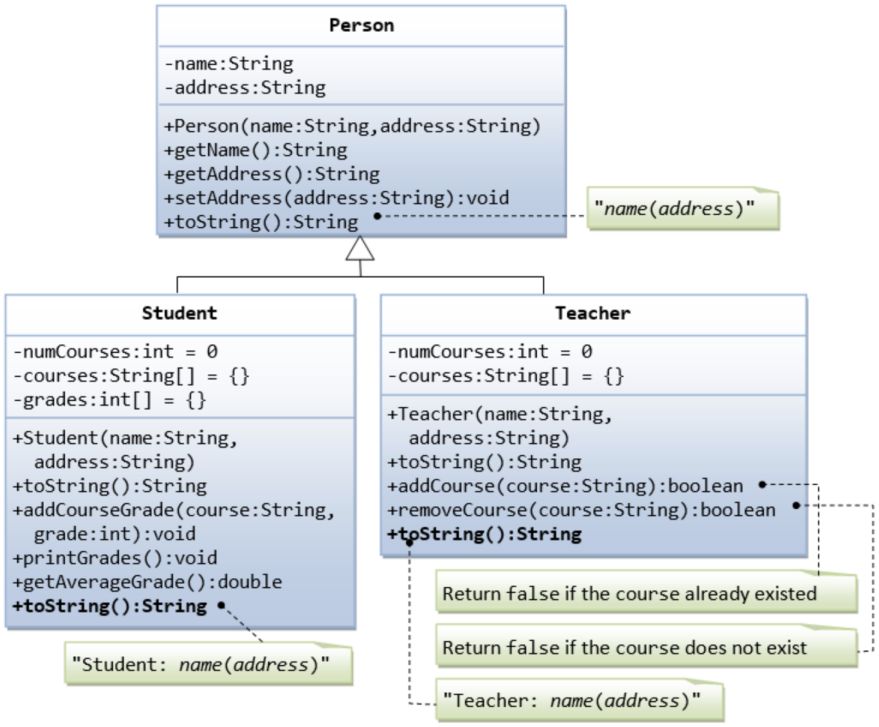
Classes are broken up into 3 compartments:

- Name
- Fields
- Methods

The class name is written in bold, and is italicised if the class is abstract. Similarly, abstract methods are italicised.
On Monday, someone asked me about representing interfaces in UML. For an interface, the name is preceded by

```
<<interface>>
```

Fields and methods are marked as private **-**, public **+**, or protected **#**.
I'll return to that UML diagram that we worked with on Monday.



3.2 Relationships

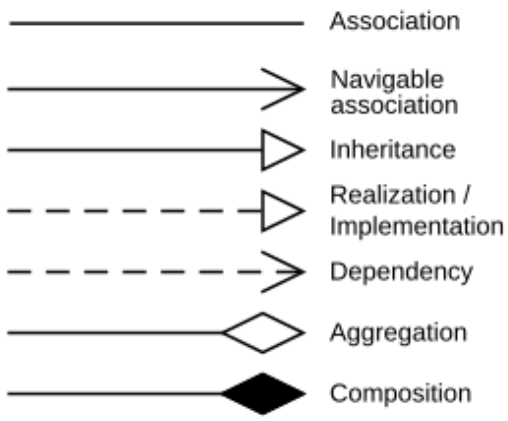


Figure 1: https://en.wikipedia.org/wiki/Class_diagram

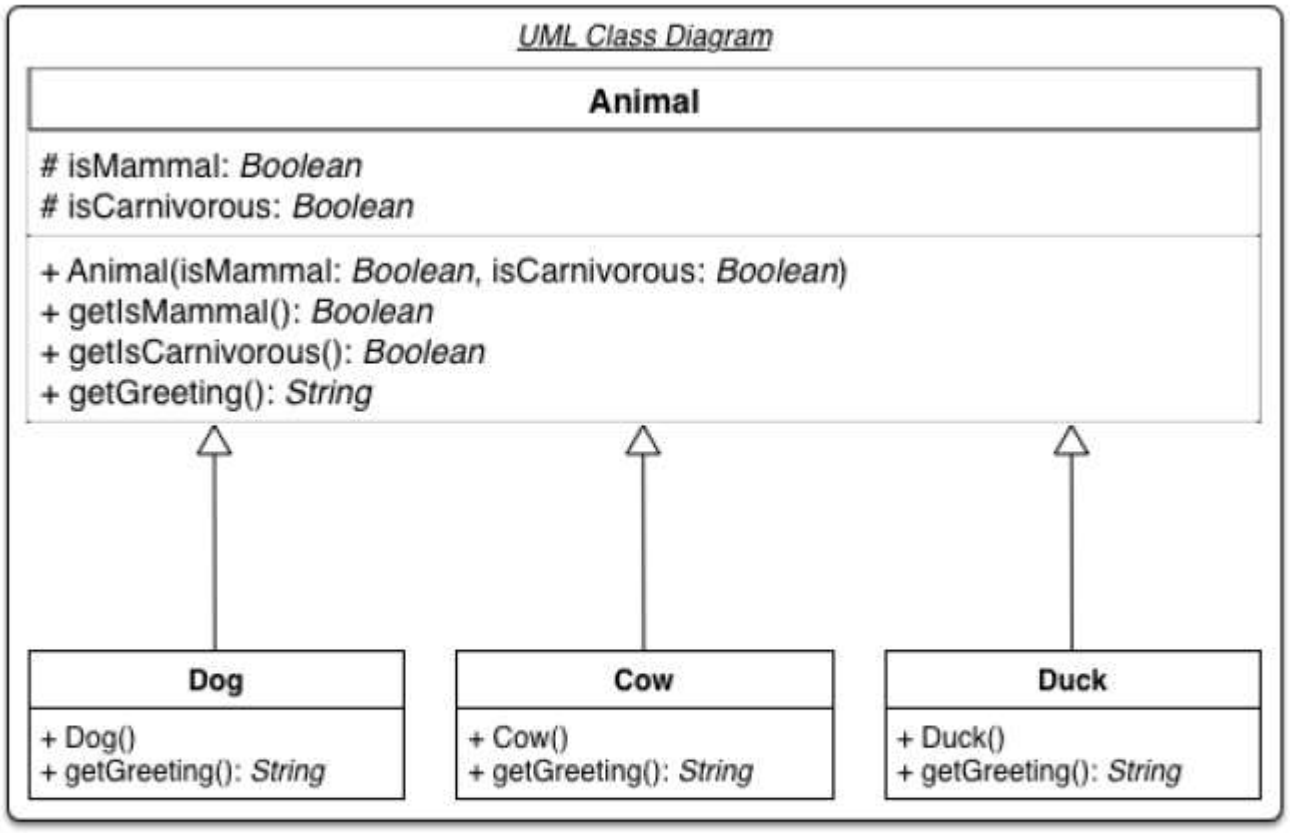
- An association indicates that two classes must communicate with one another. A line with no arrows represents a bidirectional association (e.g., a Student and Teacher relationship).
- We saw an example of inheritance above. This relationship points from the subclass to the superclass (i.e., from the child class to the parent class).
- Realisation/Implementation is used when a class implements an interface or an abstract class
- "Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist." https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

- "Composition implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House." https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

4 Programming Problem

I have taken this problem from the tutorial 3 slides up on Moodle. Start by recreating this UML diagram in Moodle, and adjust to account for Animal being an abstract class (italicise where needed and adjust the relationship notations!). Also add some methods and fields to Dog, Cow, and Duck to make them more interesting to work with. After that, start implementing the program as specified in the problem statement.

Exercise 4: Consider the following UML diagram:



A UML diagram of Animal, Dog, Cow, and Duck classes. Recall that - denotes *private*, + denotes *public*, and # denotes *protected*.

Things to do:

1. Declare an abstract class named Animal with the implementations for getIsMammal() and getIsCarnivorous() methods, as well as an abstract method named getGreeting() .

2. Create Dog, Cow , and Duck objects.

3. Call the getIsMammal() , getIsCarnivorous() , and getGreeting() methods on each of these respective objects.

4. Three classes named Dog , Cow , and Duck that inherit from the Animal class.

5. No-argument constructors for each class that initialize the instance variables inherited from the superclass.

6. Each class must implement the getGreeting() method:
- For a Dog object, this must return the string ruff .
- For a Cow object, this must return the string moo .
- For a Duck object, this must return the string quack .

Input Format
There is no input for this challenge.

Output Format
The getGreeting() method must always return a string denoting the appropriate greeting for the implementing class.

Sample Output
A dog says 'ruff', is carnivorous, and is a mammal.
A cow says 'moo', is not carnivorous, and is a mammal.
A duck says 'quack', is not carnivorous, and is not a mammal.