COMP249 Tutorial 2: Intro to Object Oriented Programming

Ella Noyes

July 14, 2024

1 A Note On Java Code Conventions

As you learn new programming languages, try to get familiar with style conventions (e.g., naming conventions). Each programming language has its own set of "best practices" created to help developers write code that is clear, maintainable, and scalable. For Java code conventions, refer to "Java Code Conventions" from Oracle.

2 Classes and Objects

Classes are structures that model real-world entities, abstract concepts, and complex structures in code. A class groups together the properties and behaviours that define a concept.

You might, for example, want to represent a book in code. A book has an author, a title, a publisher, an ISBN, a page count, etc. If you only need to represent one book in code, then you could get away with storing that data using primitive data types (int, char, boolean, ...). But if you have to represent more than one book in code, then things will get messy quickly.

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

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

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public void printBookInfo() {
        System.out.println("Title: " + title +
                           ", Author: " + author +
                           ", ISBN: " + isbn);
    }
}
```

This book class has 3 data members (a.k.a. “fields” or “attributes”) and 8 methods.

Notice that Book() looks like methods that we’ve seen before. It has an access specifier “public” and a return type “Book”, but it doesn’t have a name. Book() is a special type of method known as a **constructor**. This method “contructs” instances (i.e., objects) of the book class.

2.1 The static Keyword

static methods and variables relate to the class in general. For Book: title, author, and isbn will be specific to every book object. But you may want a static variable "count" that counts the total number of Book objects created in your program.

```
public class Book {
    private static int bookCount = 0;

    private String title;
    private String author;
    private String isbn;

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

    public static int getBookCount() {
        return bookCount;
    }
        // etc. etc.
}
```

2.2 Access Specifiers

Access specifiers (a.k.a access modifiers) specify whether a class' member variables and methods may be accessed by other classes.

Access specifiers, in order of most accessible to most restrictive:

- public (class, package, subclass, global)
- protected (class, package, subclass)
- Default–when no modifier is given (class, package)
- private (class)

We'll look through examples of how/when to use different specifiers. My approach is to plan out a program before I start the implementation (using UML diagrams and some pseudocode). In planning out the entire project, you can identify the scope of a class' member variables and methods.

3 Inheritance

A class may be extended by another class. The resulting child class, or **subclass**, is said to **inherit** from its parent class, or **superclass**. This new subclass inherits variables and methods from its parent, and can have its own additional set of behaviours and characteristics. Inheritance can make code more reusable, extensible, simpler to maintain, and supports polymorphism (we will see an example of this in a moment!). The inheritance relationship is sometimes referred to as an is-a relationship. Similarly, class composition, when one class contains an instance of another class as one of its members, can be referred to as a has-a relationship

3.1 Method Overriding

When a class extends another class, it may need to adapt or specialise the methods inherited from its superclass. In Java, you can use the @Override annotation to verify that you are correctly overriding a method from the superclass. (Again, we'll see an example of this in a moment). Methods that are marked as **final** can not be overriden!

4 Final Point: Abstract Classes and Interfaces

I was having a hard time phrasing this. I got ChatGPT to generate this definition for me!

Abstract classes and interfaces are both mechanisms in Java for defining methods that can be implemented by subclasses or implementing classes, but they serve different purposes and have distinct characteristics.

- Abstract classes are used when you want to provide a common base with shared code and allow subclasses to extend it while optionally overriding methods. They can contain both abstract methods (without implementation) and concrete methods (with implementation).
- Interfaces, on the other hand, are used to define a contract for what methods a class should implement without providing any implementation details. A class can implement multiple interfaces, enabling a form of multiple inheritance.

Overall, abstract classes are best for creating a foundational class with shared behavior, while interfaces are ideal for specifying capabilities that can be shared across different class hierarchies.

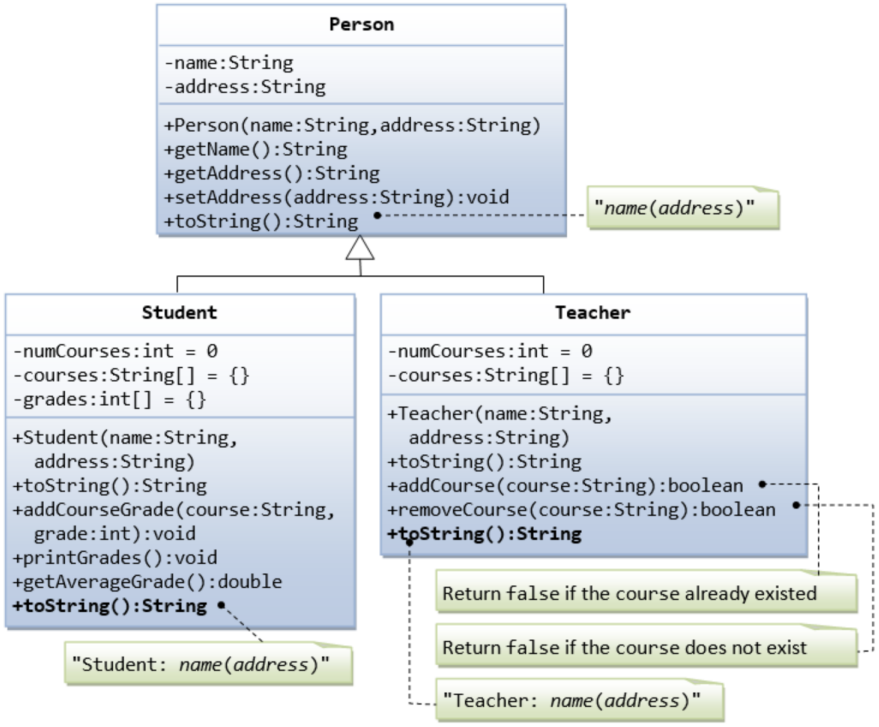
5 Live Coding

I'll show you an implementation covering all of these concepts.

6 A Programming Problem

I browsed the world wide web for some inheritance practice problems. I wanted to find one that included a UML diagram to get you comfortable with starting from there. The following problem was created by Chua Hock Chuan from Nanyang Technological University. Find more Class Composition, Inheritance, and Polymorphism problems from them here: https://www3.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html

"Suppose that we are required to model students and teachers in our application. We can define a superclass called Person to store common properties such as name and address, and subclasses Student and Teacher for their specific properties. For students, we need to maintain the courses taken and their respective grades; add a course with grade, print all courses taken and the average grade. Assume that a student takes no more than 30 courses for the entire program. For teachers, we need to maintain the courses taught currently, and able to add or remove a course taught. Assume that a teacher teaches not more than 5 courses concurrently."



If you finish this before the tutorial ends, then you can practice Class Composition! Introduce a new class, "Course", that has one teacher and a list of students. Course should have a title, a course code, a value in credits, a maximum number of students, a list/array of Students, one Teacher, etc. etc. ... Add some appropriate methods to the Course class (e.g., addStudent(), assignProfessor(), printStudents(), ...), and adapt Student and Teacher classes (e.g., in Student, add a method to increment numCourses and add a course to courses[] when the Student successfully enrols in a course).