COMP249 Tutorial 4: Reviewing OOP Basics and UML Diagrams

Ella Noyes

July 20, 2024

1 Inner Classes

Inner classes are classes that are defined (and therefore contained) within a class. There are four types of inner classes. I'll show you some code demonstrating each of these in a moment.

1. **Static Inner Classes:** Since these are static, they may be instantiated without instantiating the outer class. And since they are static, they may not access non-static variables and methods from their outer class.
2. **Non-Static Inner Classes:** These may access the outer class' non-static variables and methods.
3. **Local Inner Classes:** Defined within a block (e.g., a method, constructor, or an initialization block.) It is local to that block; it can only be instantiated and used within the block where it is defined.
4. **Anonymous Inner Classes:** A type of local inner class that does not have a name and is used to instantiate an object of a class or interface with an immediate implementation.

We'll look at my example code before moving on!

2 Back to Class Composition

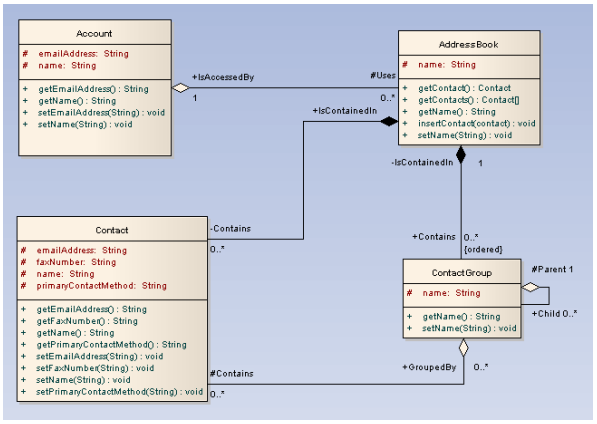

Figure 1: Diagram is from: sparxsystems.com/resources/tutorials/uml2/class-diagram.html

This UML clas diagram has 2 composition relationships (the filled diamond) and 3 aggregation relationships (the hollow diamond).
AddressBook contains 0 or many Contact objects. Because this is a composition relationship, those Contact objects are destroyed when that AddressBook class is destroyed.
In contrast, a ContactGroup object contains 0 or many Contact objects and when a ContactGroup object is destroyed, its Contact objects continue to exist independently. This hollow diamond represents an aggregation relationship.

3 Enumerations (enums)

Enums are a data type that define a set of named values that behave as constants. I showed you an example of enums in tutorial 2:

```
public class Robot {
    public enum RobotType {
        ROBOT_THERAPIST,
        ROBOT_FRIEND,
        ROBOT_LOVER,
        ROBOT_DOG,
        ROBOT_CHEF,
        ROBOT_RECEPTIONIST
    }

    private RobotType robotType;
    private String name;

    public Robot(RobotType robotType) {
        this.robotType = robotType;
        this.name = Person.generateName();
    }

    public void sayHello() {
        System.out.println("Hello my name is " + name + " and I will be your " + robotTy
    }
}
```

4 Programming Problems

4.1 Class Composition Problem

"Write a Java program to create a class called "Library" with a collection of "Book" objects and methods to add and remove books." - https://www.w3resource.com/java-exercises/oop/index.php
You can expand this problem into inheritance practice by substituting "Book" with an abstract base class "BorrowableItem" and creating concrete derived classes Book, CD, DVD, ...

4.2 Inheritance Problem

"Write a Java program to create a class called "Event" with attributes for event name, date, and location. Create subclasses "Seminar" and "MusicalPerformance" that add specific attributes like number of speakers for seminars and performer list for concerts. Implement methods to display event details and check for conflicts in the event schedule. - https://www.w3resource.com/java-exercises/oop/index.php

5 Glossary of Concepts Covered So Far

I got ChatGPT to spit out a glossary of the keywords that we've discussed so far. I have gone through that list and reworded some of it. Most of these terms have synonyms that you should be familiar with.

5.1 Basic Concepts

5.1.1 Class

Definition: A structure that models a real-world entity, an abstract concept, or a complex structure in code.

5.1.2 Object

Definition: An instance of a class, representing a specific entity with state and behavior.
Synonyms: Instance (of a class)

5.1.3 Constructor

Definition: A special method used to initialize objects.

5.1.4 Field

Definition: A variable defined within a class that holds data for objects.
Synonyms: Attribute, data member, instance variable

5.2 Inheritance and Relationships

5.2.1 Inheritance

Definition: A class may be extended by another class. The resulting derived class is said to **inherit** from its base class. This derived class inherits variables and methods from its parent, and can have its own additional set of behaviours and characteristics.
Synonyms: Extends, IS-A relationship

5.2.2 Base Class

Definition: A class that is extended by other classes, providing common attributes and methods.
Synonyms: Super Class, Parent Class

5.2.3 Derived Class

Definition: A class that extends another class (the base class) to inherit its properties and methods.
Synonyms: Sub Class, Child Class, Concrete Class (if it is concrete! This term is typically used when the class extends an abstract class and implements its abstract methods)

5.2.4 Polymorphism

Definition: From the Greek "Having multiple forms". The ability of a single method to perform different functions based on the context. It allows objects of different classes to be treated as objects of a common superclass.

5.2.5 Encapsulation

Definition: A concept that involves bundling data and methods that operate on the data into a single unit, or class. It restricts direct access to some of the object's components to prevent unintended interference and misuse of the object's internal state. Encapsulation is achieved through access modifiers that control the visibility of class members.
Synonyms: Data Hiding, Data Encapsulation

5.2.6 Interface

Definition: A reference type that defines a contract for classes to follow. It consists of abstract methods, default methods, static methods, and constants. Interfaces specify methods that implementing classes must provide, and they can also include private methods to facilitate code reuse within the interface itself.

5.2.7 Abstract Class

Definition: A class that cannot be instantiated on its own and may contain abstract methods that must be implemented by subclasses.
Synonyms: Abstract Base Class

5.2.8 Abstract Method

Definition: A method that is declared without an implementation.

5.2.9 Method Overloading

Definition: Defining multiple methods with the same name but different parameters.
Synonyms: Compile-Time Polymorphism

5.2.10 Method Overriding

Definition: Providing a specific implementation of a method defined in a superclass.
Synonyms: Runtime Polymorphism

5.2.11 Static Keyword

Definition: A keyword that indicates methods and variables that relate to the class in general.

5.2.12 Final Keyword

Definition: A keyword used to define constants, prevent method overriding, and inheritance.

5.3 Design Principles

5.3.1 Composition

Definition: A design principle where a class is composed of one or more objects from other classes, implying a has-a relationship.
Synonyms: Has-a relationship

5.3.2 Aggregation

Definition: A specialized form of composition representing a whole-part relationship where the part can exist independently of the whole.

5.4 Modifiers and Keywords

5.4.1 Access Modifiers

Definition: Keywords that specify the visibility of classes, methods, and fields.
Synonyms: Access Specifiers, Visibility Modifiers