COSC 3P91 – Advanced Object-Oriented Programming
Laboratory 3: UML-to-Code Structural Mapping

Department of Computer Science
Brock University

Winter Term

Overview

This laboratory serves as a bridge between architectural design and technical implementation. Students are required to demonstrate a full understanding of Unified Modeling Language (UML) by translating a complex class diagram into a functional Java project structure. The exercise emphasizes the precision required to map various relationship types—composition, aggregation, association, and dependency—into standard Java syntax.

Learning Objectives

By completing this laboratory, students will be able to:

- Interpret complex UML 2.0 notation, including visibility modifiers and multiplicity.
- Implement structural relationships: Composition (lifecycle binding), Aggregation (independent existence), and Dependency (usage).
- Realize interfaces and extend abstract classes to enforce system-wide contracts.
- Translate static utility signatures and factory patterns from design to code.

Given Problem Description

Students are provided with the `SmartHome_ClassDiagram.png`, which represents the version 2 architecture of the "SmartHome Pro" system. This model includes high-level interfaces for device control, abstract foundations for hardware and sensors, and specialized concrete implementations for home automation logic.

Analysis Tasks

Before coding, students should analyze the diagram to identify:

1. Which relationships necessitate the use of the `final` keyword or constructor-based instantiation to ensure lifecycle binding?
2. Which associations require the use of Collections (e.g., `List`, `Set`) to handle 1..* or 0..* multiplicities?
3. How to represent the difference between an `implements` and an `extends` relationship in Java.

Required Task

The sole task for this laboratory is to **construct a Java implementation** based strictly on the provided UML class diagram. The implementation must follow these constraints:

- **Structural Fidelity**: Every class, interface, and abstract class shown in the diagram must be present in the code.

- **Relationship Accuracy**: Relationships (Composition, Aggregation, Inheritance) must be correctly implemented via fields, constructors, or inheritance keywords.
- **Visibility and Members**: All attributes and methods must use the correct visibility modifiers (+/−/#) and data types specified.
- **Method Stubs**: Implement the signature of every method. *Note: You do not need to implement the logic/bodies of the methods.*

Key Takeaways

Structural integrity is the foundation of maintainable software. By accurately mapping a design to code, developers ensure that the intended object lifecycles and behavioral contracts are preserved throughout the development process.