June,2023

Department of Computer Science,
Faculty of Architecture and
Engineering

LECTURER

Igli Draci

STUDENT

Xhesi Baze
Antea Toska

Composition Over Inheritance

A Software Paradigms project

Table of Contents

I Background information on Inheritance and Composition

Inheritance:

Inheritance is a mechanism in object-oriented programming where a class (referred to as a child class/subclass) inherits properties and behaviors from another class (referred to as its parent class/superclass). The child class can extend or modify the functionality inherited from the parent class while also adding its own unique features.

Key points about inheritance:

- Inherited properties and behaviors are accessible in the child class.
- The child class can override inherited methods to provide its own implementation.
- Inheritance promotes code reuse and enables the hierarchical organization of classes.

Composition:

Composition is a design principle that allows objects to be composed of other objects as parts or components. Instead of inheriting behavior, an object can have a reference to another object and pass on some of its tasks to that object.

Key points about composition:

- Composition emphasizes building complex objects by combining simpler objects.
- Objects can be composed of one or more other objects.
- Composition promotes code modularization, flexibility, and reusability.

II Downsides of Inheritance

In most cases, inheritance is a concept Computer Science students grasp early on and choose to stick to while solving the majority of programming problems they deal with. Contrary to popular belief, the usage of inheritance is not encouraged by experts and there are a few reasons behind this.

- **Inheritance can lead to fragile software.**

Inheritance is safe to use within a package, where the subclass and the superclass implementations are done by the same developers. Inheritance is safe to use when extending a class that is specifically designed to be extended by a specific class. In case of inheriting outside of the same package, the software may break and perform undefined behavior.

- **Inheritance is not flexible.**

A subclass' implementation depends on the details of its superclass. This, not only violates **encapsulation**, it also makes the relationship between a subclass and its superclass highly important. The superclass implementation may need to change over time, and if it does, the subclass needs to change by the superclass. This may not sound that bad when dealing with small programming problems. However, when we have to deal with large software, this option is not practical, as well as time and money-consuming.

III Concrete example

HTTPS://GITHUB.COM/XHESBAZE/COMPOSITIONOVERINHERITANCE

IV Who to use when?

The Liskov Substitution Principle

The LSP states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. While not directly prioritizing composition over inheritance, it implies that:

- Inheritance is preferred to be used when we encounter an "is-a" relationship between classes, where a subclass is considered a specialized version of the superclass.
- Inheritance is preferred to be used when the subclass is also a subtype of the superclass.
- Real-life examples: An apple IS a fruit; a tiger IS an animal; a motorcycle IS a vehicle.

- Composition is preferred to be used when we encounter an "has-a" relationship between classes.
- Composition is preferred to be used when the subclass is not a subtype of the superclass.
- Real-life examples: A human HAS a heart, a person HAS a degree, an animal HAS a habitat.