# CMPT 225: Data Structures & Programming – Unit 02 – Object Oriented Design, Part One

Dr. Jack Thomas
Simon Fraser University

Spring 2021

## Today’s Topics

- Definition of Object-Oriented Design
- The Four Principles
- Inheritance
- Polymorphism
- Design Patterns

## What Does It Mean To Be "Object Oriented"?

- A **paradigm** for organizing code into discrete "objects", each complete and self-contained.
- Adopted by many of the **most popular programming languages**, including Java, C++, and Python.
- Provides a useful **framework for understanding** data structures and algorithms that is transferrable between different languages.

# The Four Principles

1. **Abstraction**: Summarizing complicated systems into their overall concept.
2. **Encapsulation**: Containing whatever you need to fulfill that concept within the object.
3. **Modularity**: Making each of these objects distinct and reusable.
4. **Hierarchy**: Organize your objects into a hierarchy of is-a relationships.

## 1. Abstraction

- The process of **organizing code around a clearly defined purpose** it’s meant to fulfill.
- Applying abstraction to data structures gives us **Abstract Data Types (ADT)**, meaning the abstract idea of a data structure like a stack or queue rather than the specific implementation.
- The point is to make programming easier for humans to understand and talk about with each other.

# A Word On Objects, Classes, and Instances

- Abstraction makes it necessary to distinguish between different **instances** of the same **class** of object.
- For comparison, every individual dog in the world is an instance of a “dog”, the concept.
- An array of ten Strings is also a set of ten objects, instances of the String class.
- Understanding how individual instances are **unique but also alike** is the key to understanding object-oriented programming.

## 2. Encapsulation

- An object should contain **everything it needs** to complete its purpose, and **nothing more**.
- It should also only take in what it needs, and only put out what it's supposed to, **hiding its inner workings**.
- The programmer is **free to implement this code however they want**, so long as the object behaves as expected.

# How does a car work?

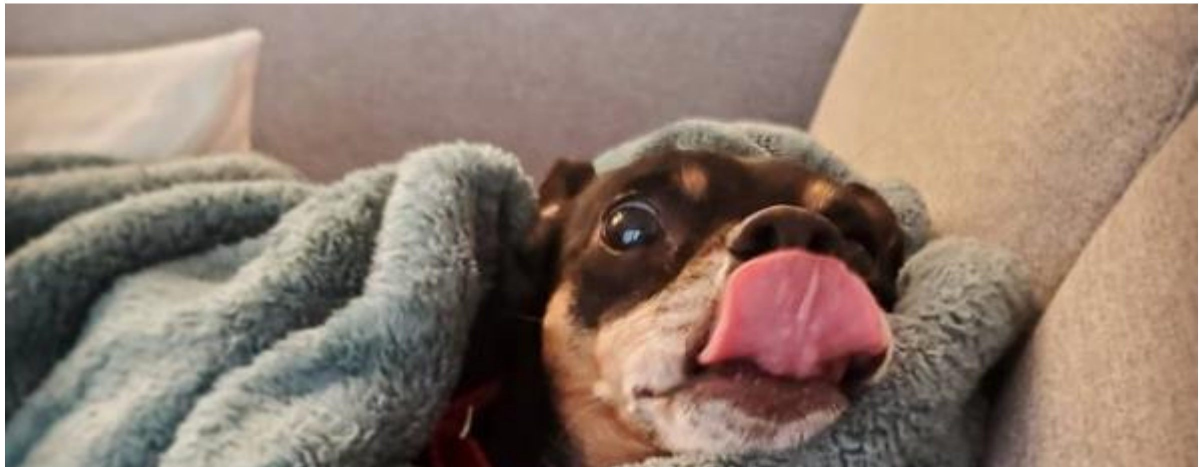
- **Answer**: You don’t need to know, so long as the pedals and steering wheel do what they’re supposed to.

## 3. Modularity

- Along with giving each object a clear purpose and making them self-contained, they should also be **distinct** from one another.
- Each object should only be concerned with doing its own job, while the programmer focuses on arranging them in the right order.
- This makes it easier to **reuse code** to solve similar problems - objects should behave the same if dropped into a new environment.

## 4. Hierarchy

- Objects are typically organized into a hierarchy of is-a relationships according to their type (e.g. Chihuahua is a Dog).
- More specific objects descend from more general ones, but belong to both (e.g. a Chihuahua named Fido is both a Chihuahua and a dog).



# Object-Orientation in Java

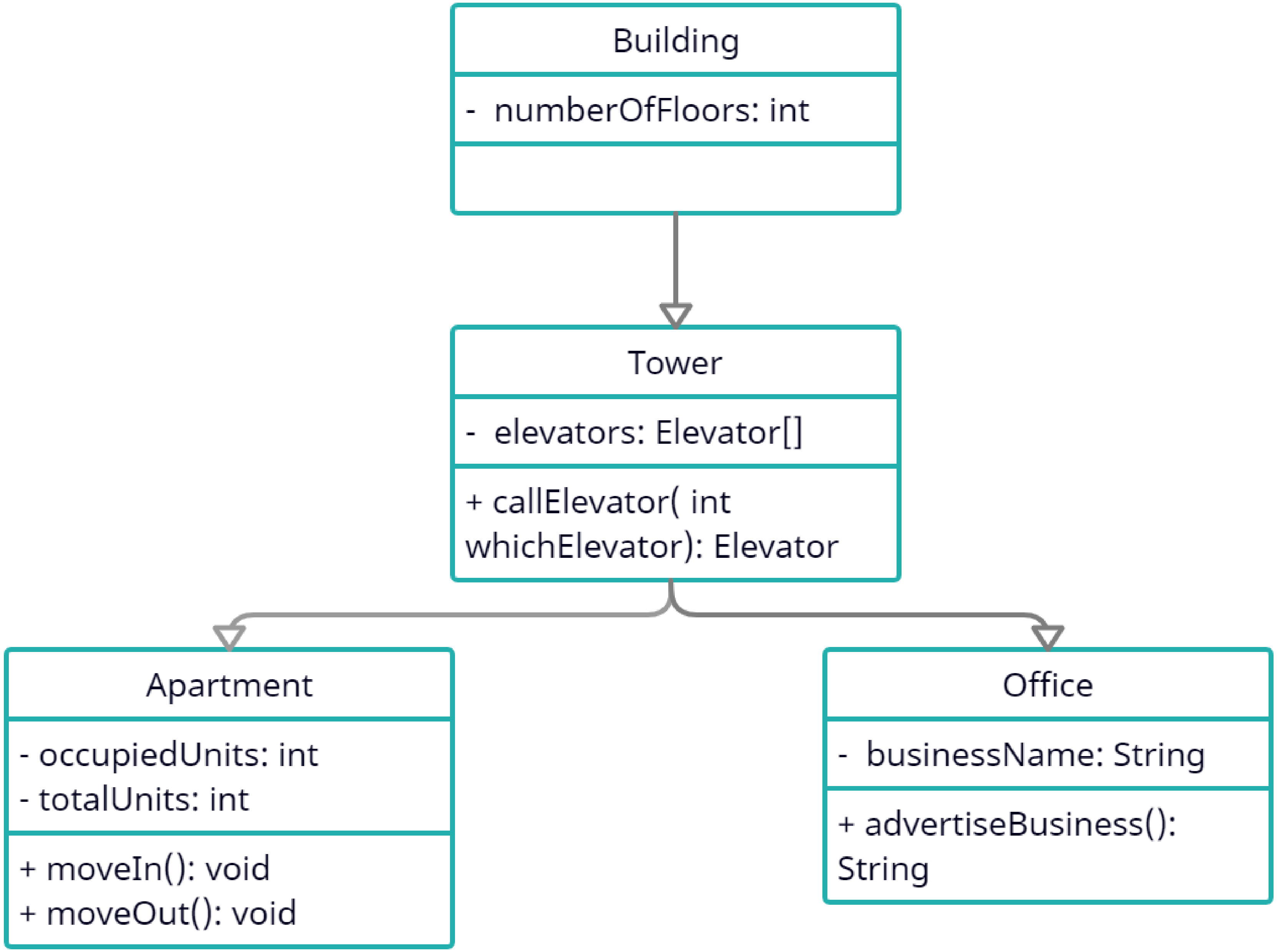
- In Java, code objects are grouped together according to their **class**, with general **superclasses** (or **base classes**) at the top and specialized **subclasses** extending from them.
- The **standard library** contains many modular packages for you to use, including **interfaces** that act as Abstract Data Types.
- Good Java programming follows these examples.

# Inheritance

- When objects are organized in a hierarchy, the more specific objects are said to **inherit** from their more general predecessors, gaining their **fields** and **methods**.
- When done well, a programmer should **only need to write code for the newest and most specific parts** of their new classes that inherit from the existing general cases, cutting down on **redundancy**.

## Inheritance in Java

- Subclasses inherit the **variables** (fields) and **functions** (methods) of their superclasses.
- Note that classes can only inherit from (extend) one other class, but that multiple classes can inherit from that class.
- A subclass inherits everything from their superclass’s superclasses as well, creating a **chain of inheritance**.


Building
- numberOfFloors: int
Tower
- elevators: Elevator[]
+ callElevator( int whichElevator): Elevator
Apartment
- occupiedUnits: int
- totalUnits: int
+ moveIn(): void
+ moveOut(): void
Office
- businessName: String
+ advertiseBusiness(): String

## Polymorphism

- Literally “many forms”.
- Hierarchy and inheritance creates situations where different objects can fulfill the same purpose – if you need a dog, both a Chihuahua and a Daschund would work.

## Polymorphism in Java

- **Overriding** is when a subclass implements its own version of a function belonging to its superclass. Java defaults to the version of a function defined by the lowest subclass.
- **Overloading** is when a class has more than one version of the same function, distinguishing between them with different **signatures** (e.g. what variables they accept as their argument).

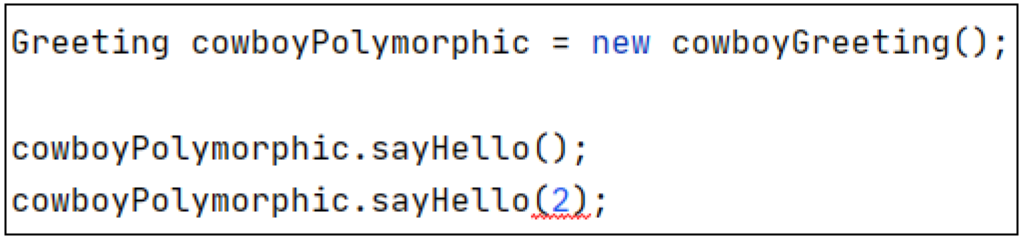
```java
class Greeting
{
    public void sayHello()
    {
        System.out.println("Hello!");
    }
}

class cowboyGreeting extends Greeting
{
    public void sayHello()
    {
        System.out.println("Howdy partner!");
    }
    public void sayHello(int numberOfPartners)
    {
        System.out.println("Howdy, my " + numberOfPartners +" partners!");
    }
}
```

```java
Greeting example = new Greeting();
cowboyGreeting cowboyExample = new cowboyGreeting();

example.sayHello();
cowboyExample.sayHello();
cowboyExample.sayHello( numberOfPartners: 2);
```

```
Hello!
Howdy partner!
Howdy, my 2 partners!
```



```
Greeting cowboyPolymorphic = new cowboyGreeting();

cowboyPolymorphic.sayHello();
cowboyPolymorphic.sayHello(2);
```

The first line will print “Howdy, partner!” but the second line will not compile, because the program doesn’t know that cowboyPolymorphic is a cowboyGreeting.

## Design Patterns

- Best practices and common solutions to problems in object-oriented programming can be grouped together into a **design pattern**.
- Design patterns are like blueprints for a kind of program, structure, algorithm, etc, that are usually general and not tied to one language.
- Common features of a design pattern include a **name**, a **context** where they're used, a **template** for implementing them, and a **result** that describes their output and performance.

## Some Design Patterns We May Cover

**Algorithms**

- Recursion
- Amortization
- Divide and Conquer
- Prune and Search
- Brute Force
- Greedy Method
- Dynamic Programming

**Software**

- Position
- Adapter
- Iterator
- Template method
- Composition
- Comparator
- Decorator

# Recap – The Summary Pattern

- **Object-oriented programming** is a popular programming paradigm that informs how programmers approach code across multiple languages.
- It follows principles of **abstraction**, **encapsulation**, **modularity**, and **hierarchy** to organize code.
- In Java, **subclasses inherit** methods and fields from their **superclasses**, and can replace them through **overriding** or make alternatives through **overloading**.
- **Design patterns** are blueprints for making certain common software elements according to best practices.