Object-Oriented Design

Section 2.1

Object-Oriented Design

- A method of program design in which the main elements are objects.
- Supported by using an object-oriented language like C++ or java.
- An object is an instance of a class, which specifies the type of data members any instance contains, as well as member functions (also called methods) that any instance can execute.
- A class should present a concise and consistent interface to the objects that are instances of the class.
- Class interfaces should not go into unnecessary detail about the inner workings of their instances. This is called information hiding.

Object-Oriented Design Goals

- Correctness
- Robustness
- Adaptability
- Portability
- Reusability
- Readability
- Security
- Parallelizability

Object-Oriented Design Principles

- Abstraction
- Encapsulation (information hiding)
- Modularity
- Hierarchical Organization

Abstraction

- Abstraction is to distill a complicated system down to fundamental parts and describe these parts in simple, precise language.
- Describing an abstracted system involves naming the fundamental parts and describing their variation or functionality.
- Systems can have many different abstractions derived from them. The usefulness of an abstraction depends on the task at hand.
- Applying abstraction to the design of data structures gives rise to abstract data types (ADTs).

Abstract Data Types (ADTs)

- An ADT is a mathematical model that specifies the general type of the data that is stored and the operations that are permitted on the data (along with the types of the parameters of the operations).
- An ADT specifies what each operation does, but not how it does it.
- The functionality of an ADT is expressed in C++ as the public interface of the class representing the ADT.
- A class is said to implement an interface if its functions include all the functions declared in the interface, and possibly more.

Encapsulation

- Encapsulation is the idea that different components of a software system should not reveal the internal details of their implementations.
- Also called information hiding.
- Encapsulation aids programmers in establishing program correctness and giving freedom when implementing the details of a subsystem.

Encapsulation Example .h file

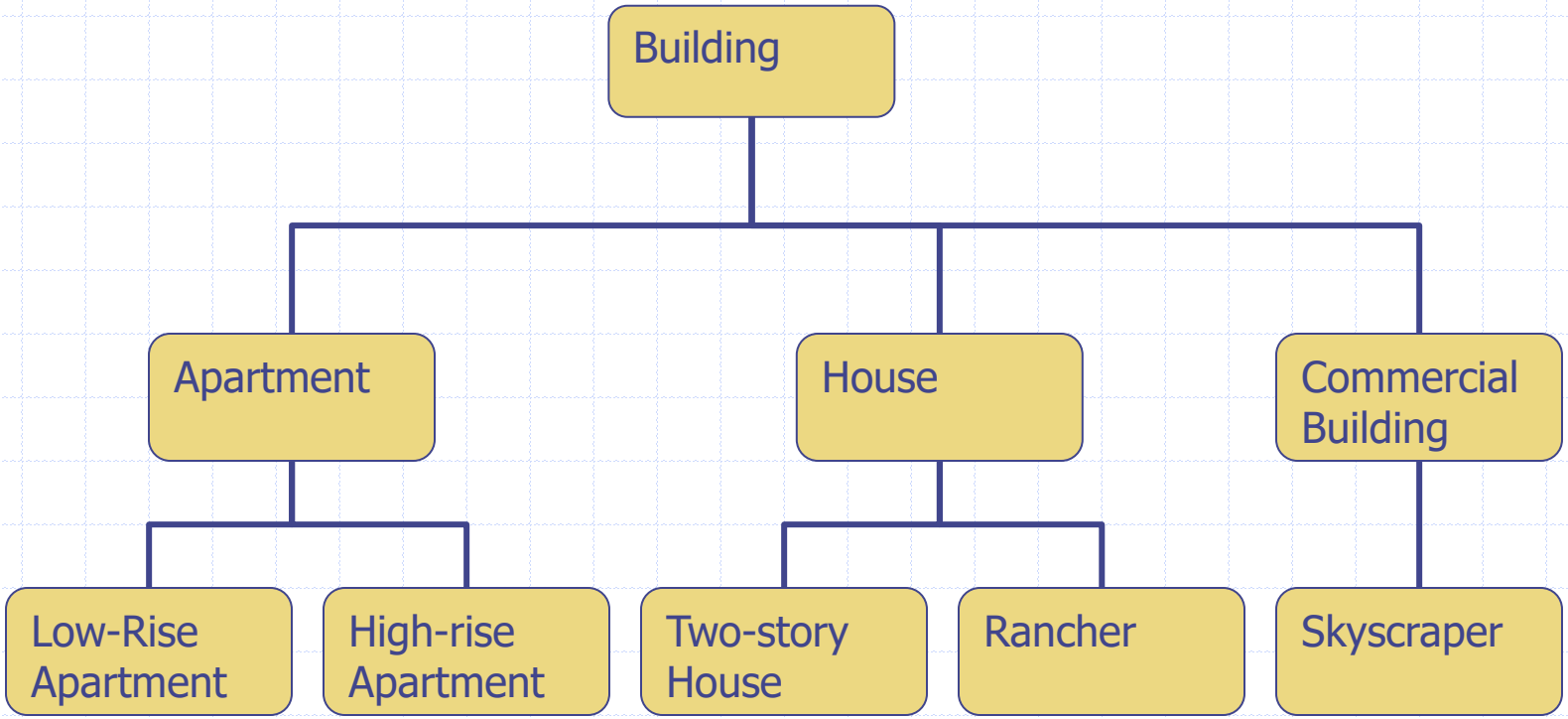
```
class Point {
  public:
    static Point *makeCartesian(double x, double y);
    static Point *makePolar(double r, double theta);

    double getX();
    double getY();
    double getR();
    double getTheta();
}
```

Modularity

- Modularity refers to organizing code so that different components of a software system are divided into separate functional units.
- For example, we could have a game with a database component, a networking component, a user-interface component, and a game logic component. Each component is kept separately in different modules.
- Modularity helps with software reusability.

Hierarchical Organization

- IS-A hierarchies.



Design Patterns

- A design pattern has a name and is a template for a solution to a problem in a context.
- They are "best practices" for algorithm or software design.
- We will encounter algorithm design patterns and software design patterns in this course.

Algorithm Design Patterns

The book covers

- Recursion
- Amortization
- Divide-and-conquer
- Prune-and-search
- Brute force
- The greedy method
- Dynamic programming

But we will probably not see the whole list in class.

Software Design Patterns

The book covers

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

And we've already seen

- Factory method