Introduction to OOP and UML

- Programming Paradigms
  - Procedural
  - Functional
  - Object Oriented
- Object-Oriented Design
- Unified Modeling Language
  - Use Cases
  - Class Diagrams
  - Sequence Diagrams
- **Reading:**
  - L&C, App A
  - **http://algs4.cs.princeton.edu/11model**

Programming Paradigms

- A paradigm is an ideal or a model to follow in doing something, e.g. programming
- In ***CS110***, you were introduced to the three predominant programming paradigms:
  - Procedural
  - Functional
  - Object-Oriented
- In this course, we will further explore the Object-Oriented paradigm using ***Java***

Object-Oriented Design

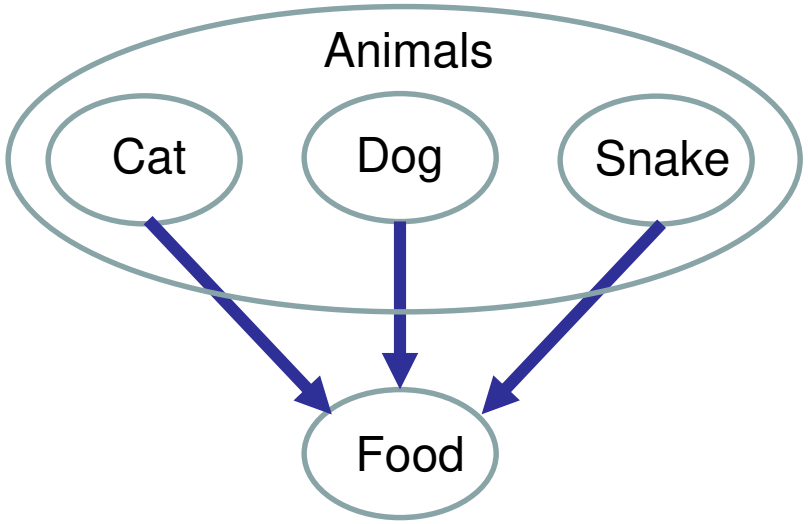
- The **Object-Oriented Programming (OOP)** paradigm was developed by software engineers to solve most if not all of the problems described in *L&C, Section 1.1*.
- This indicates that software designed according to OOP principles treats data as objects, belonging to classes.
- It has become a predominant programming style for use in many applications, e.g. graphical user interfaces (GUIs).
- Java is considered to be an OOP language because it has specific features to fully support OOP

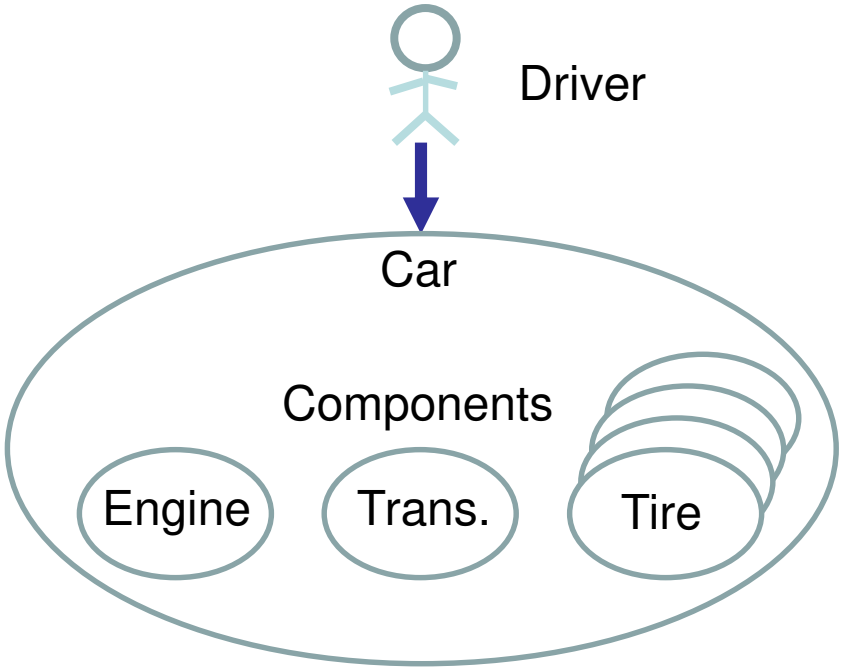
Object-Oriented Design

- In the OOP paradigm, the designer focuses on the *data*, rather than the algorithmic steps
- The data is associated with *objects* that are present in the problem, such as:
  - Monitor
  - Keyboard
  - File
  - Frame for a GUI
  - Icon in a frame for a GUI

Object-Oriented Design

- Use UML "use case diagrams" to sketch out aspects of the problem you are trying to solve
- Don't get too detailed yet





- Do NOT use flow charts or pseudo-code yet

Object-Oriented Design

- "Classify" the objects based on similarities of their attributes and behaviors
  - Examples of objects that could be classified as **Animals**: *Dogs*, *Cats*, *Snakes*, *Fish*, etc.
- Identify objects that have other objects as components or parts of themselves
  - Example: A **car** is made up of an *engine*, a *transmission*, and *tires*
- Identify objects that use other objects
  - A `Driver` drives a `Car` and an `Animal` eats `Food`

Object-Oriented Design

- Define the *names* of classes for the objects
  - Usually **nouns**, e.g. `Driver`, `Car`, `Animal`, `Cat`, etc.
- Define the *attributes* of the classes
  - Attributes: "things that objects of the class are"
  - Usually **adjectives**, e.g. color, size, furry, etc.
- Define the *behaviors* of the classes
  - Behaviors: "things that object of the class can have done to them or services they can perform"
  - Usually **verbs**, e.g. drive, eat, etc.

Object-Oriented Design

- Define the relationships between classes
  - **Inheritance** – More specific classes from general ones, e.g. a `Cat` is an `Animal`, a `Dog` is an `Animal`
  - **Interface** – A standard way that these objects can connect to other objects, e.g. power cord, etc.
  - **Aggregation/Composition** – Classes that are built up from or composed of component classes, e.g. a `Car` has an `Engine`, a `Transmission`, and `Tire`s
  - **Dependency** – Need for objects of other classes, e.g. a `Driver` drives a `Car`, a `Cat` eats `Food`

Object-Oriented Design

- Define the **methods** of each class to support the *interactions* and *behavior* of its objects
- Identify any methods that are *identical* to methods other classes also provide to connect to a *standard interface*, e.g. implementing a network connector or a power cord on a PC
- *Flow charts* or *pseudocode* may be used for individual method designs (procedural code)

Unified Modeling Language (UML)

- The **Unified Modeling Language (UML)** was developed in the 1990's to support OOP software design
- The "three amigos" (Grady Booch, Ivar Jacobson, and Jim Rumbaugh) unified their separate methodologies when they formed Rational Corporation
- A good reference is *UML Distilled, 3rd Ed.*, Martin Fowler, Addison-Wesley/Pearson

Unified Modeling Language (UML)

- UML is a *graphical* tool to visualize and analyze the requirements and do design of an object-oriented solution to a problem
- Three basic types of diagrams:
  1. **Use Case Diagram** (Shown previously)
  2. **Class Diagram** (The most useful one for us)
  3. **Interaction Diagram**
- I will use Class Diagrams in presenting the design for our Java programs / projects

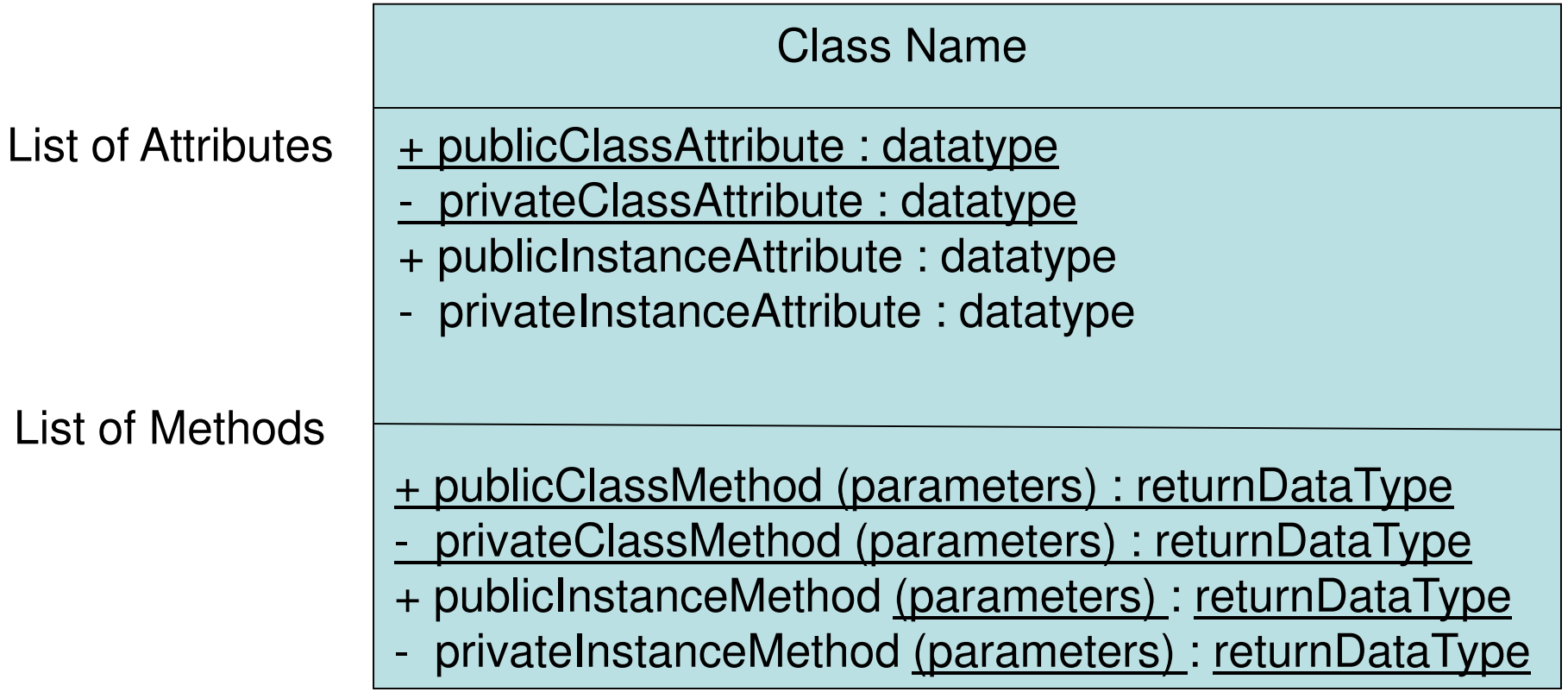
Unified Modeling Language (UML)

- Advantage of UML – It is graphical
  - Allows you to visualize the problem / solution
  - Organizes your detailed information
- Disadvantage of UML – It is graphical
  - Can be done with pencil and paper – tedious!
  - Commercial UML S/W tools are expensive!
    - Example: Rational ROSE
    - IBM acquired Rational and the three amigos got rich
  - There are some free-ware UML Design Tools

UML Class Diagrams

- UML class diagrams show:
  - The external and internal *attributes and methods* for each class
  - The *relationships* between the classes
- They're a static view of the program structure
- They do **not** show:
  - The *number* of objects of each class instantiated
  - The *timing* of the interactions between objects

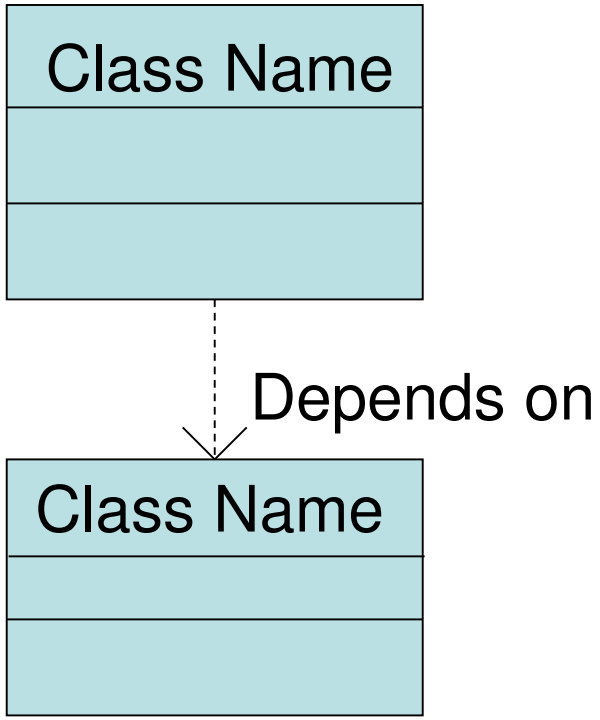
UML Class Diagrams



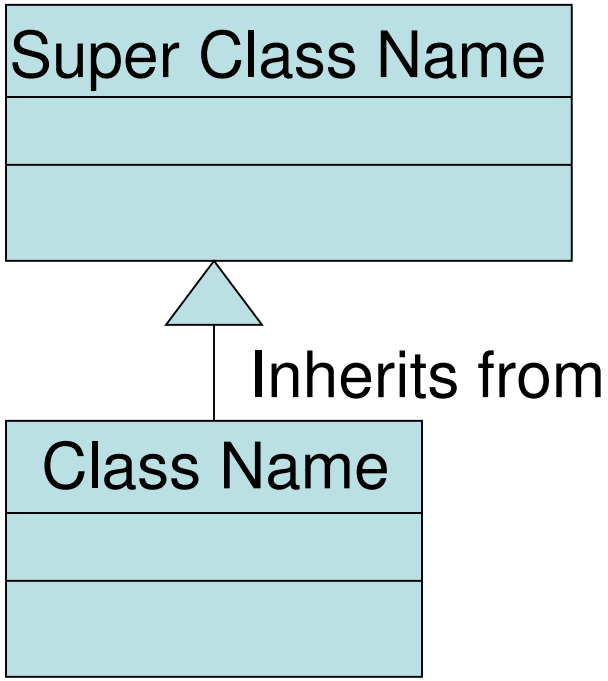
UML Class Diagrams

- UML Attribute Descriptions
  - *Protection*: + public, - private, # protected
  - Attribute *name* :
  - Attribute *data type*
- UML Method Descriptions
  - *Protection*: + public, - private, # protected
  - Method *name*
  - Method *parameter list* (name : datatype, etc.) :
  - Method *return type*
- Underlined means *class attribute or method*

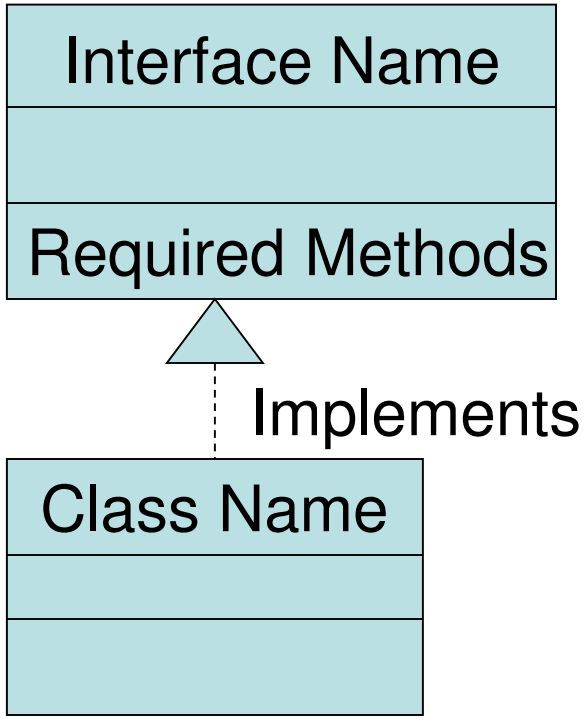
UML Class Diagrams



Driver Class
depends on
Car Class

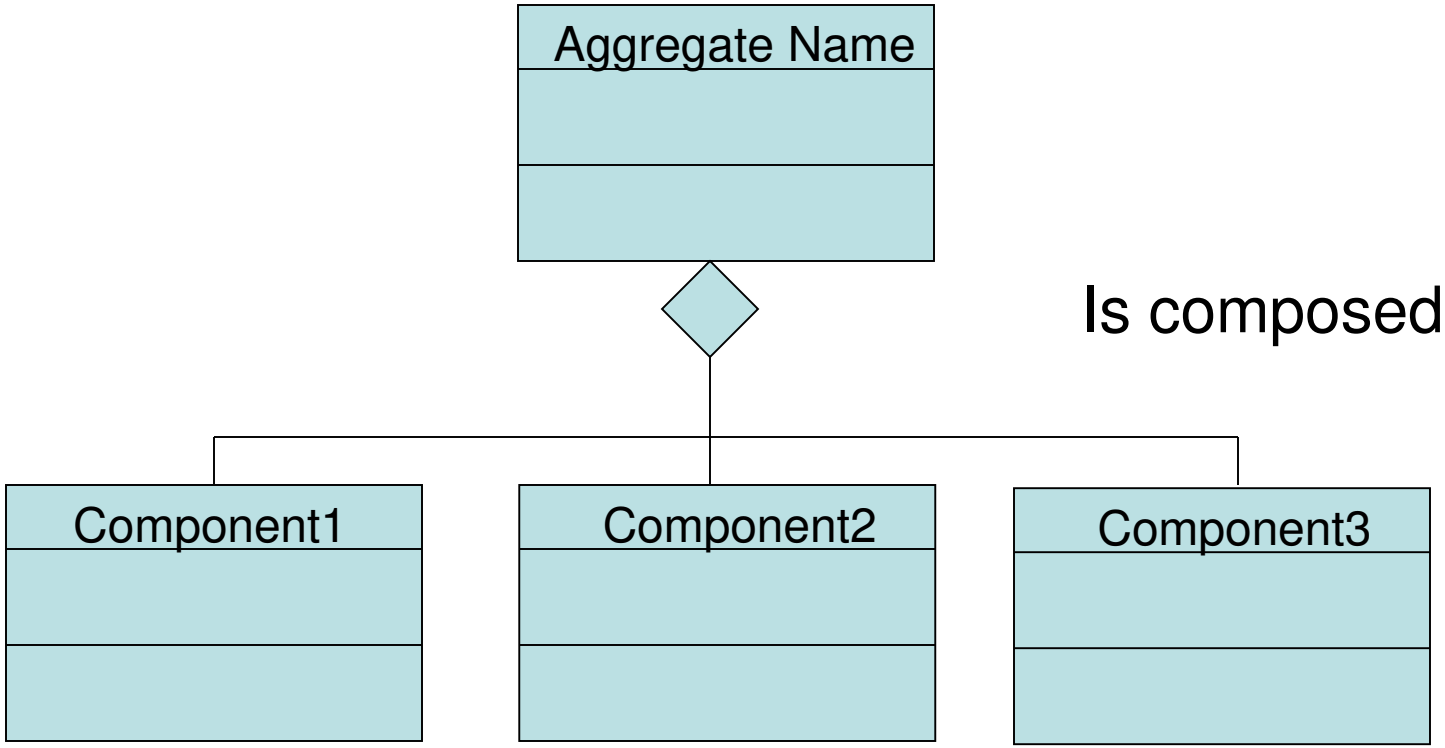


Cat Class
inherits from
Animal Class



Computer Class
implements
WiFi Interface

UML Class Diagrams



Is composed of

Car Class
is composed of
Engine Class, Transmission Class, and Tire Class

UML Interaction Diagrams

- UML interaction diagrams show
  - The *objects of each class* involved in a scenario
  - The *order of interactions* between the objects
- They are a **`dynamic`** view of the behavior
- Often called **`ladder diagrams`**, due to their resemblance to a ladder or group of ladders

UML Interaction Diagrams

A Timeline For Each `Class` : Object Involved