Introduction to Object Oriented Design and Development

2020 Spring

Review: Model

Review: Models

- Ideal Models
  - Describe a process
  - Describe relationship between objects
- Algorithmic Models
  - Describe steps to a result
  - Describe cases need to handle

Review: Why Models

- Manage complex system through abstractions
- Represent abstract concept through diagrams
- Models helps
    - Systematic testing of an entity before building it
    - Communication between stakeholders
    - Visualize abstract ideas
- No single correct model — only adequate and inadequate

Review: Benefits of Modeling

Symbols in which to express ideas

A language with which to share ideas

A controlled environment for failing

Blueprint for construction

A foundation for success

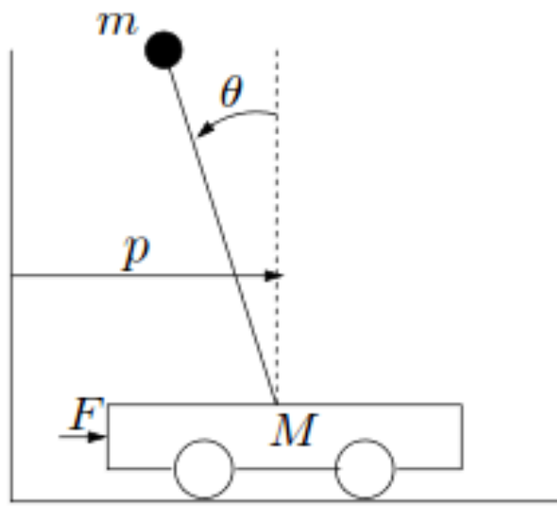
Review: Multifaceted of Modeling

- The choice of what models depends on the problem to solve and the solution will be based on.

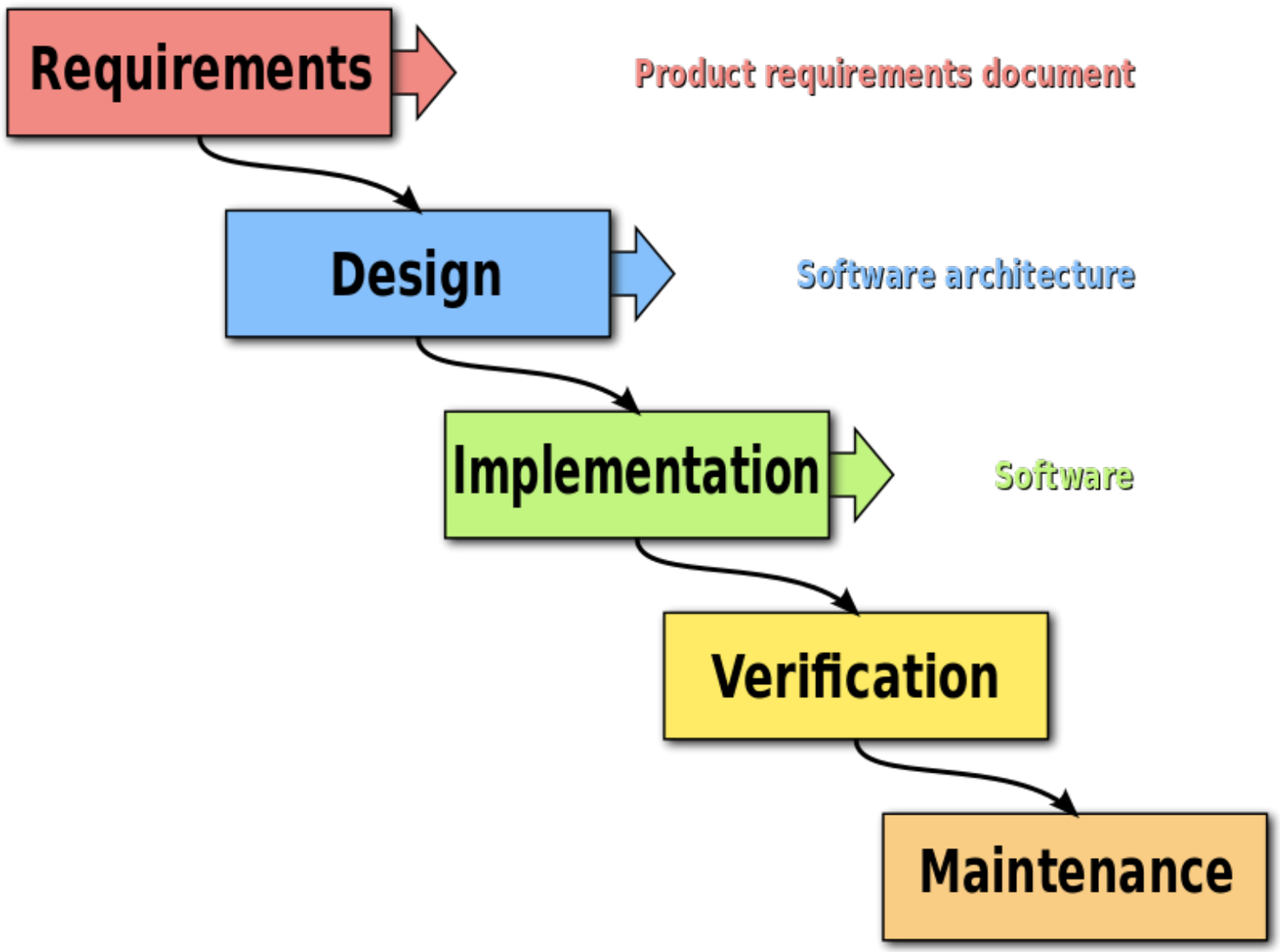
- Model can be built on different level of details.

- The good model reflects reality.

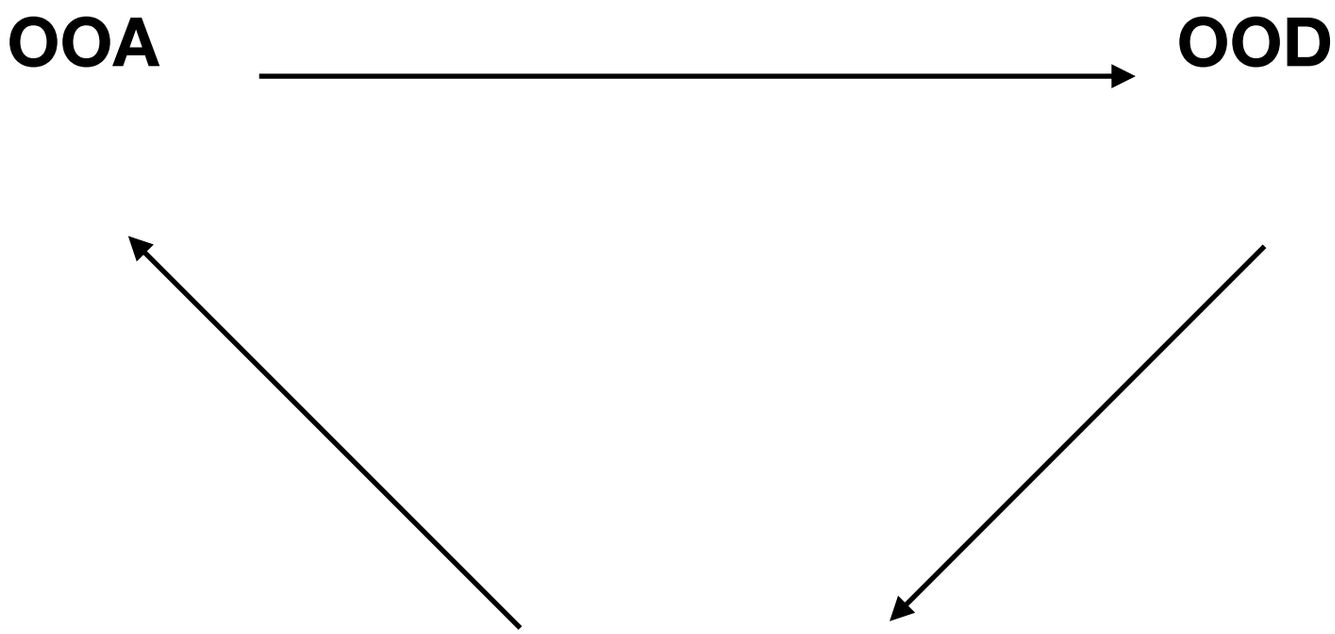
- No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.



Waterfall model



Object-Oriented Model



Class and Object

Object-Oriented History

- 1965
    - Simula programming language
- 1966
    - Alan Kay introduces OO in GUI
- Early 1970
    - Alan Kay is a leader at Xerox Palo Alto Research Center (PARC)
    - Created smalltalk

Object-Oriented Model

- Object-oriented modeling and design is a new way of thinking about problem using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity.

    - James Rumbaugh, Object-Oriented modeling and Design

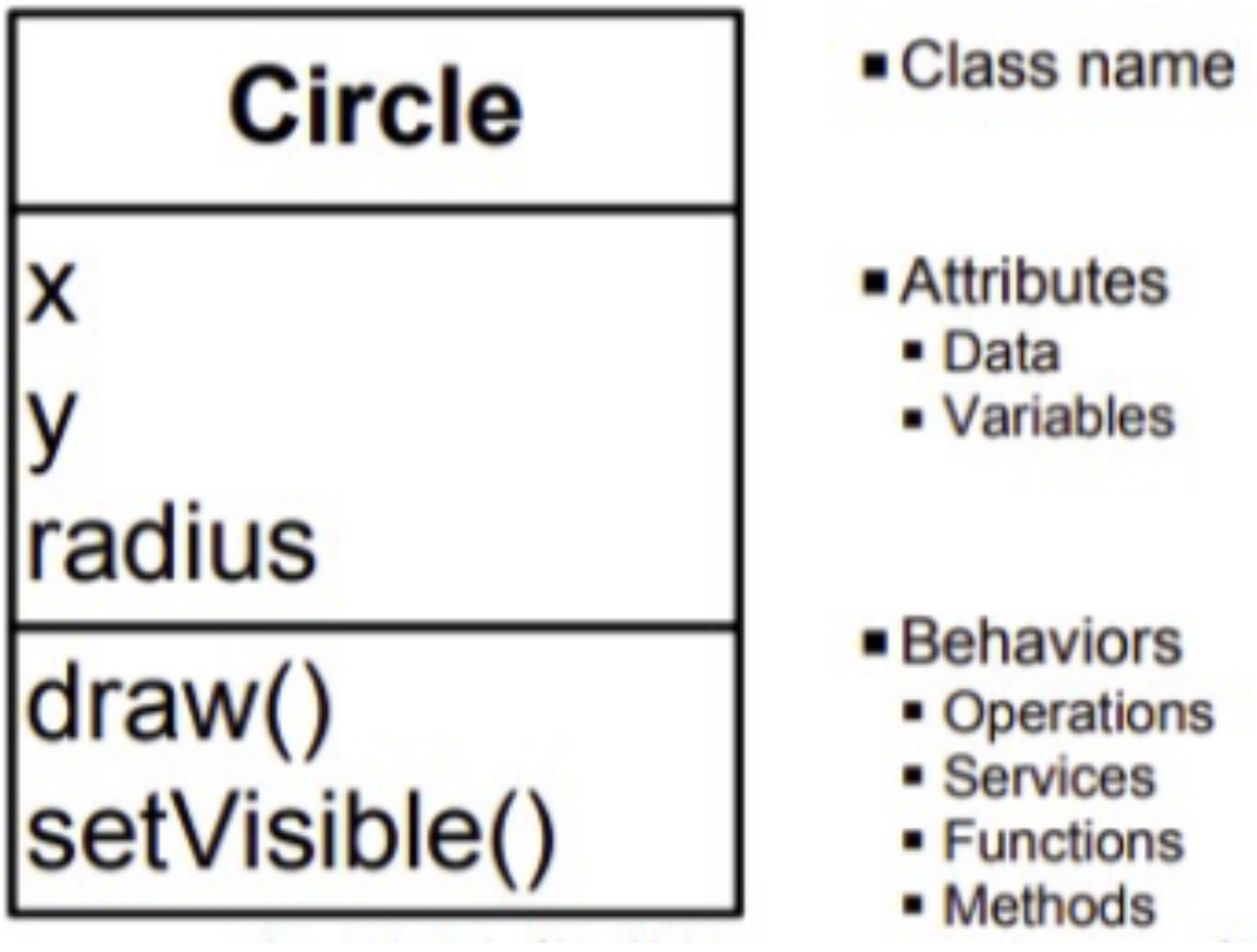
- Data structure

    - variable, instance field data, data field, data member, instance variable, state,

- Behavior/Operation

    - method, member function, function, service, sending a message is equivalent to calling a method.

Graphical Class Representation by UML



Class

- Provides data hiding
  - data is in a unique scope and access is controlled
  - They control what is exposed
    - Accessed through public interface
- Implements Abstract Data Types (ADT)
  - Creates a new type specifier
- Template, blueprint, or cookie cutter

Object

- An entity that corresponds to something in the real world
- An entity that has responsibilities
    - data, operations, send message
- Boundaries
    - has an interafce
    - limits interior access, implement data hiding
- Identifiable existence or lifetime/instance
- Identity/name

Attributes

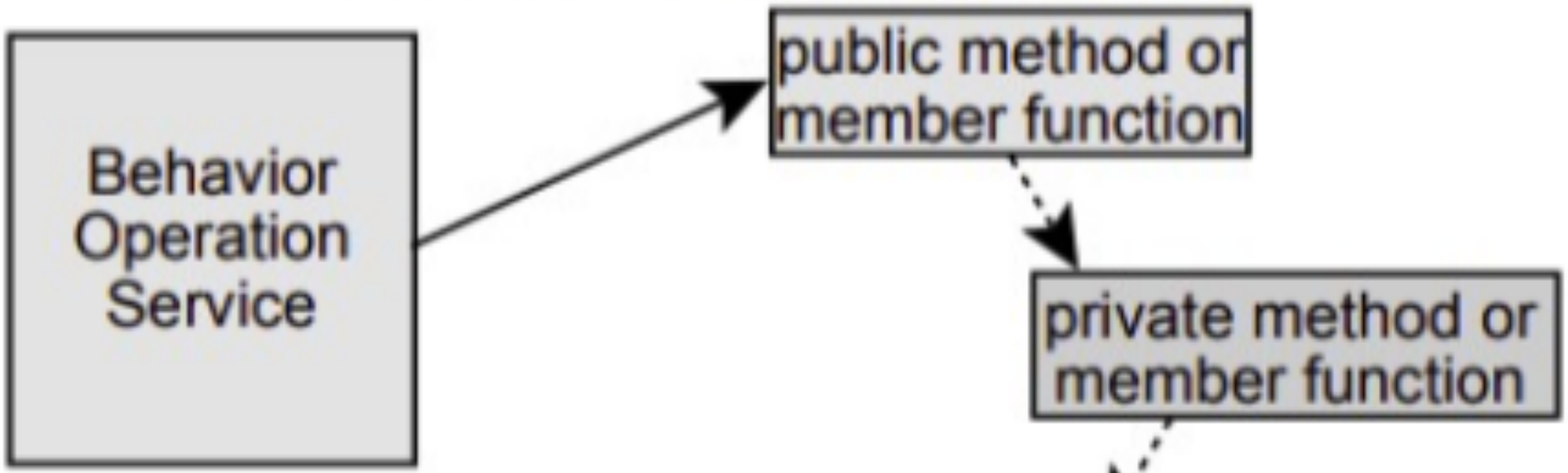
- Characterize or distinguish an object
- Are a data values held by objects
- Are the data an object is responsible for maintaining
- Are often nouns associated with the object
- Should be placed at the highest level in an inheritance hierarchy where they remain applicable to each descendant
- Good attributes depend on what is being modeled
- Derived attributes are determined from the values of other attributes.

Behaviors

- A transformation that may be applied to or by an object
- Behaviors, operations, and services are
- logical and high-level
- user visible

- Member functions and methods are
- the physical implementation of behaviors, operations, services
- May or may not be user visible
- Called though an object; that object is an implicit target
- Polymorphic operations may be applied to many classes and are implemented through multiple functions or methods

Implementing Behaviors

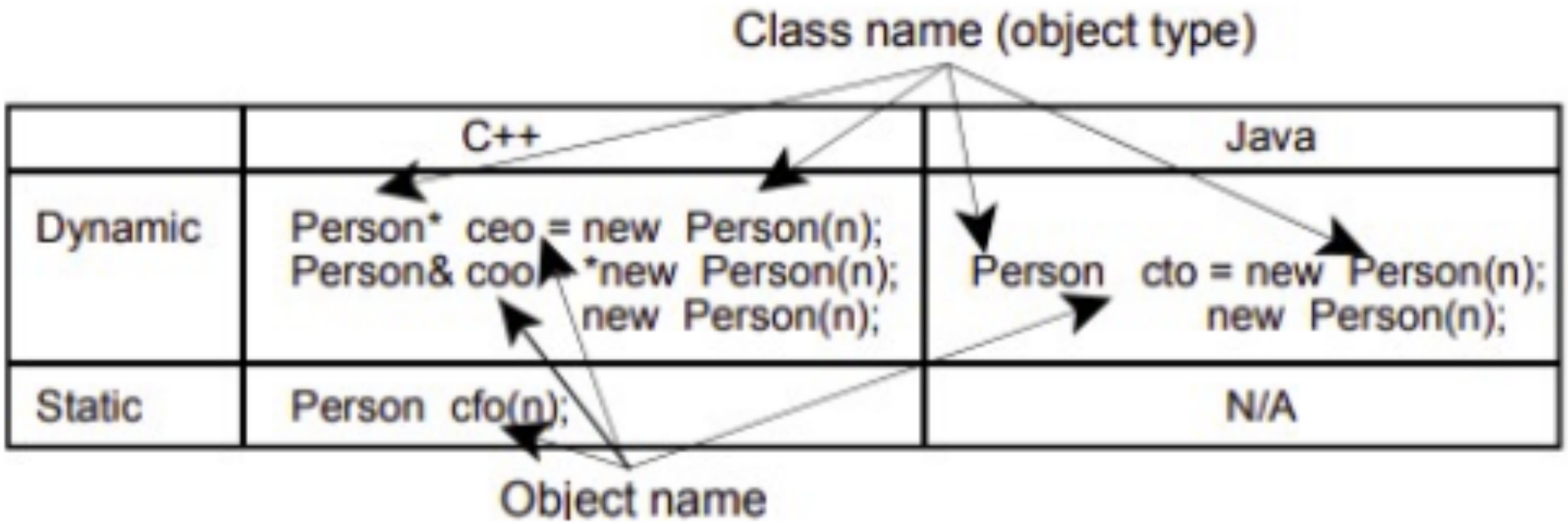


- Public interface
  - High-level (i.e., problem domain)
  - May map one-to-one with functions
  - May map one-to-many with functions

- Implementation (i.e., code)
  - public functions may call private functions

Class vs Object

- A class is a data type
- An object is a variable
    - An instance of a class
- It usually has a name.

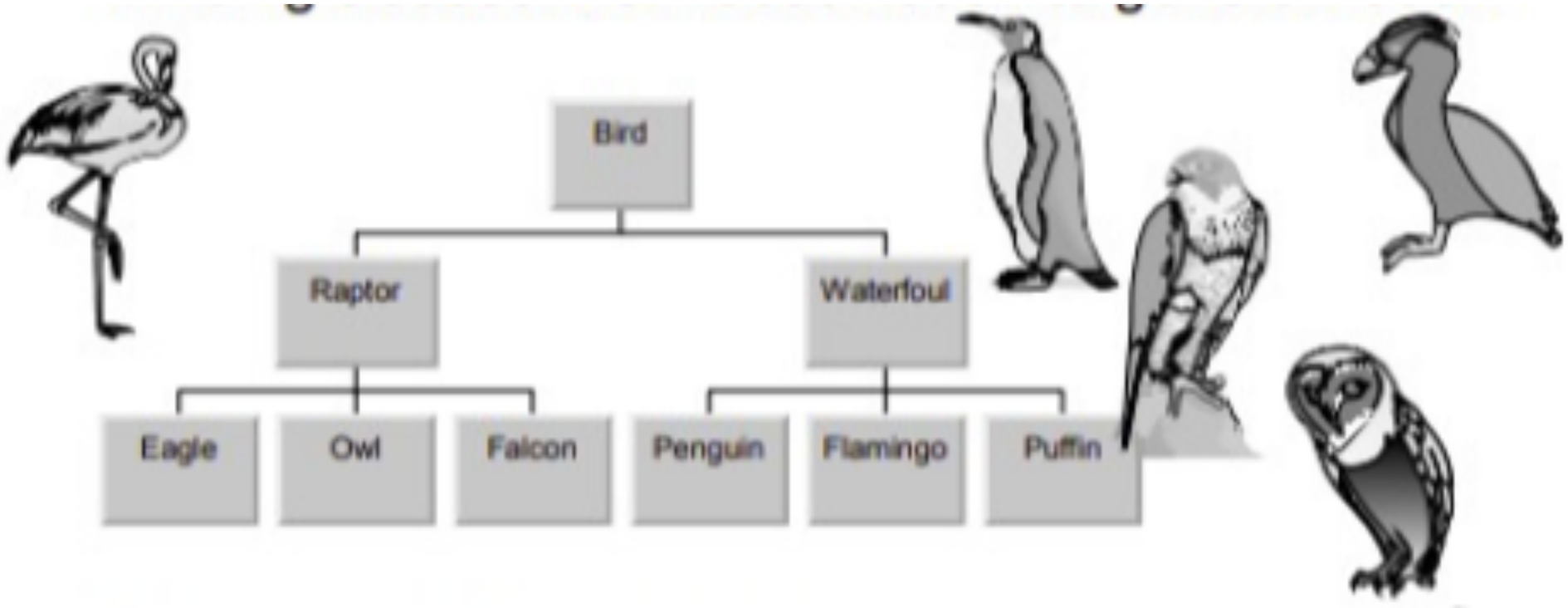


Object Oriented Concepts

Abstraction

- Abstraction
  - identifying the essential features in the problem domain
  - Separating or filtering essential features from extraneous detail
- What is essential and what is extraneous
  - depends on the problem
  - best approach depends on the object’s “position” in the program.

Classification

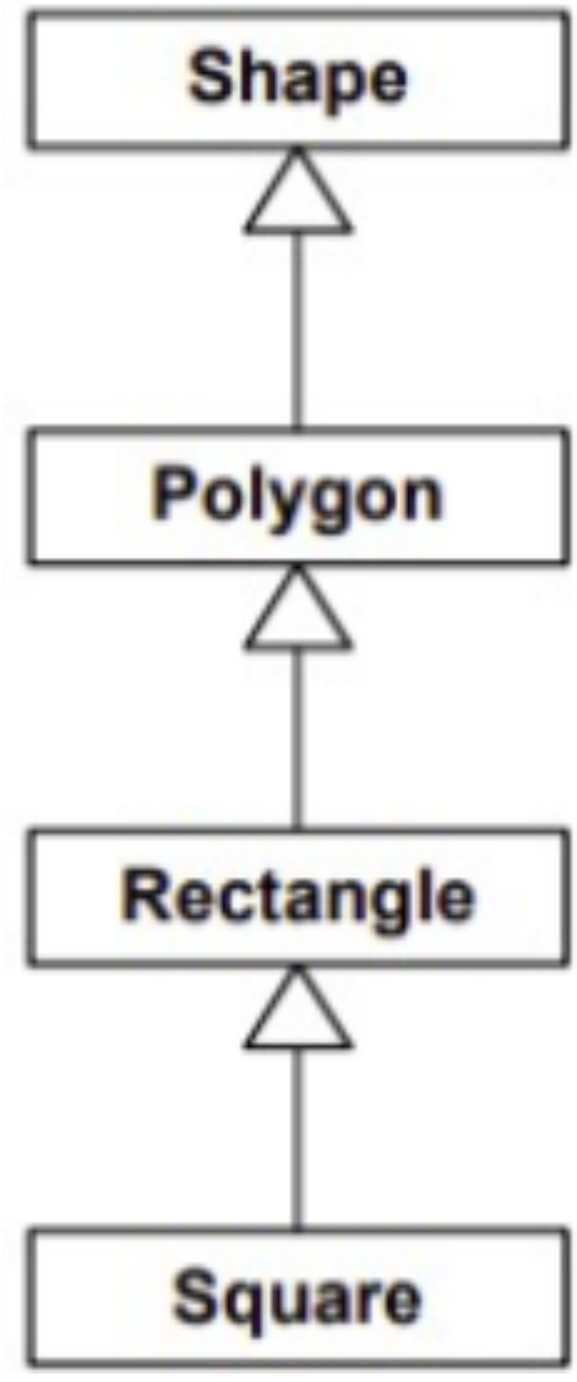


- The real world contains objects not classes
- We observe objects and classify them based on attributes, behaviors, and relationships
- Classes are artificial and arbitrary.

Class Relationships

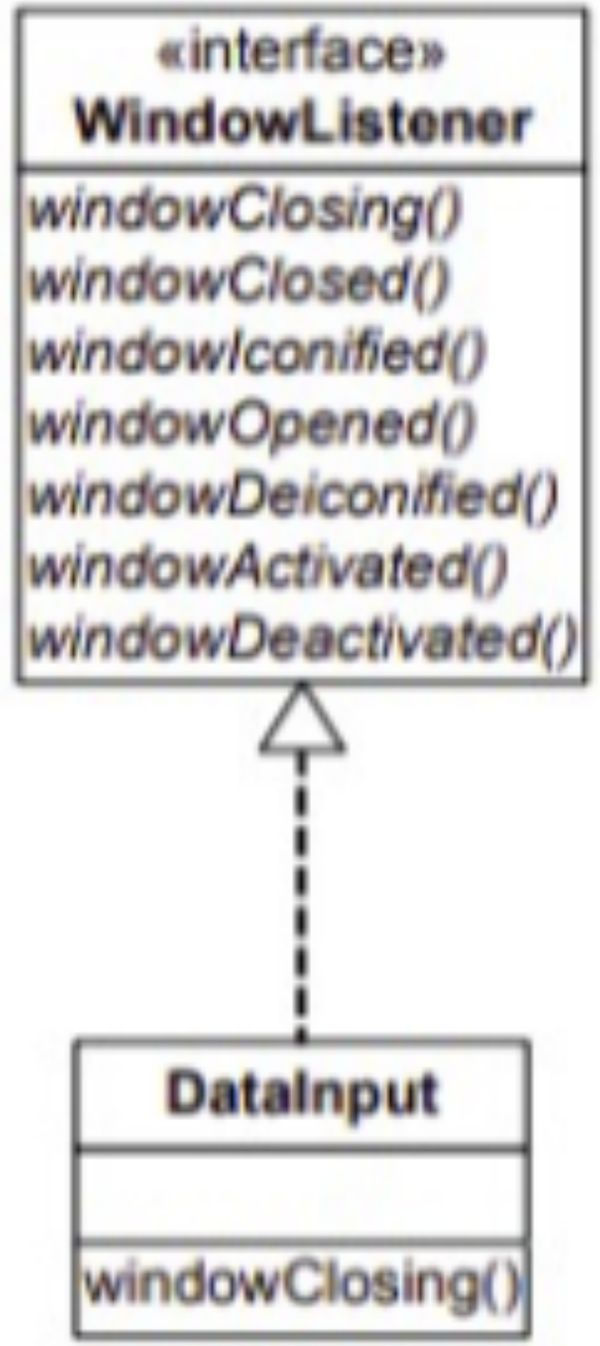
- Relationships between classes
  - promotes reuse (instructions and data)
  - Allows objects to cooperate or collaborate
- Mesages are sent along relationship lines
- Two basic relationships
  - oo features: generalization/specialization, also known as inheritance
  - Composition/Aggregation

Inheritance



Realization/Implementation

- An interface is a special class
- defines function/method signatures and return value types
- it does not have attributes
- it does not define bodies
- Realization is like inheritance except that is done with interfaces.

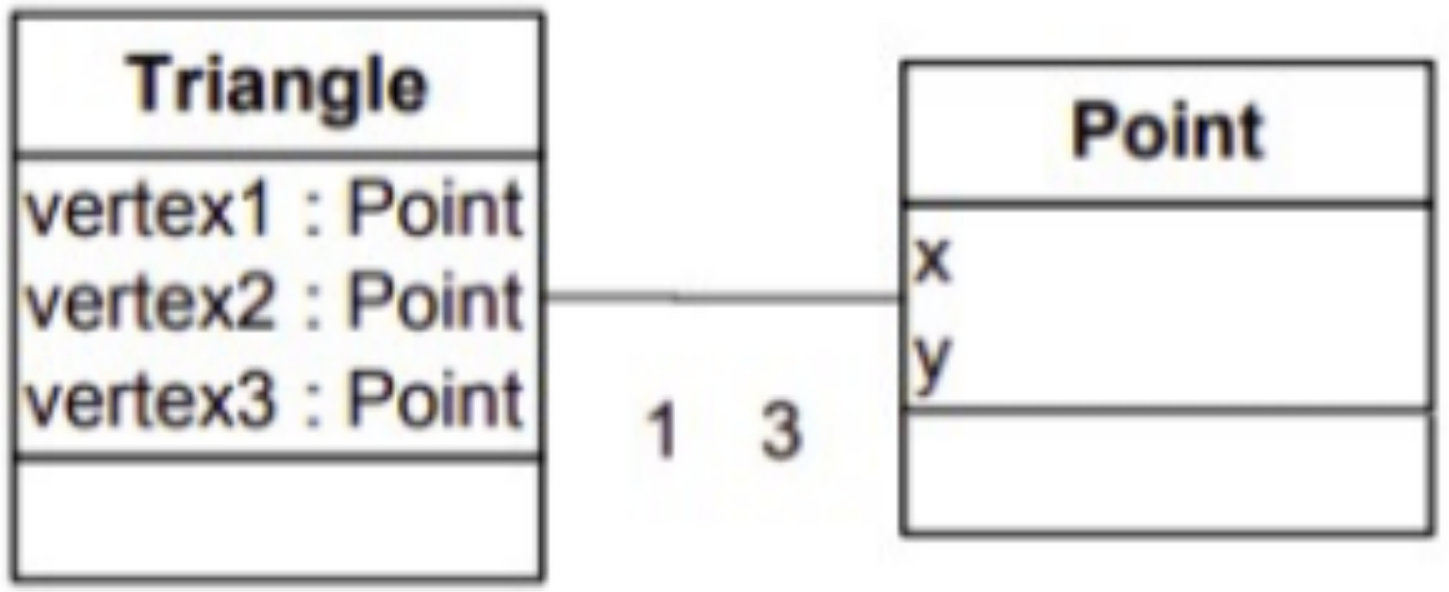


Composition

- Builds large, complex classes out of smaller, more simple classes

- Connections may have multiplicity or cardinaltiy

- Implemented with attributes



Message

- The message sent to an object must corresponds to a method of an object.

- The complete message includes

    - Object reference

    - Method name

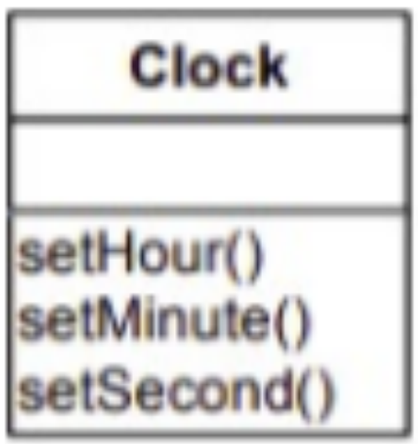
    - required data given to the object to be used by the method

- The method name and list of required arguments are what is needed to interact with the object

- The signature and return type for each method are the object's interface.

Sending a Message

```
Clock     c1(8, 32, 10);                  //C++
Clock*    c2 = new Clock(4, 58, 19);      //C++
Clock&    c3 = *new Clock(2, 0, 45);      //C++
Clock     c4 = new Clock(11, 5, 17);      //Java
```



object reference--
name of receiving object

method name--
name of message

arguments

```
c1.setHour(6);
c2->setMinute(38);
c4.setMinute(40);
c4.setSecond(52);
```

Polymorphism

- Normal situation: Compiler determines which method function to call or bind to

- Polymorphic situation: Compiler is unable to determine which method/function to call; binding is deferred until the program executes

- You know exact type when you call its method.

Defining Object-Orientation

- Encapsulation
- packaging data and operations together
    - Synonymous with an object
    - Implements data hiding
- Inheritance
    - Reusing in subclasses gain features defined in a super class
    - Object-based systems support objects but not inheritance
- Polymorphism
    - Determine at run-time which function/method to call
    - Letting an object determine which function / method is appropriate.

Extra

Constructors

- Initialize new objects

- All attributes should be explicitly initialized by the time that the constructor finishes and before the programmer is able to utilized the object

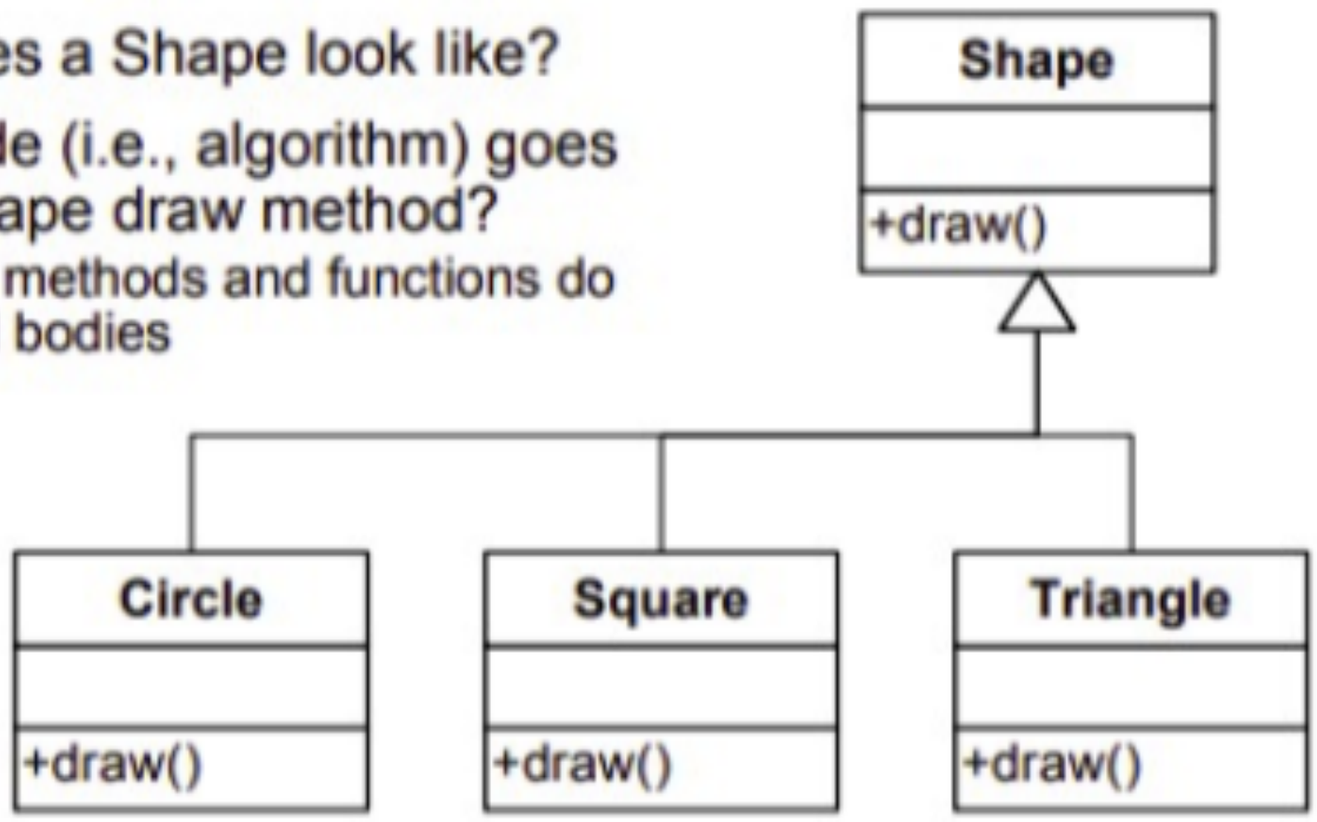
- Don’t rely on auto-initialization in the languages, may be a surprise in the end.

- Constructors may perform complex operations or they may simply initialize attributes with values passed in through the constructor argument list

Abstract Classes

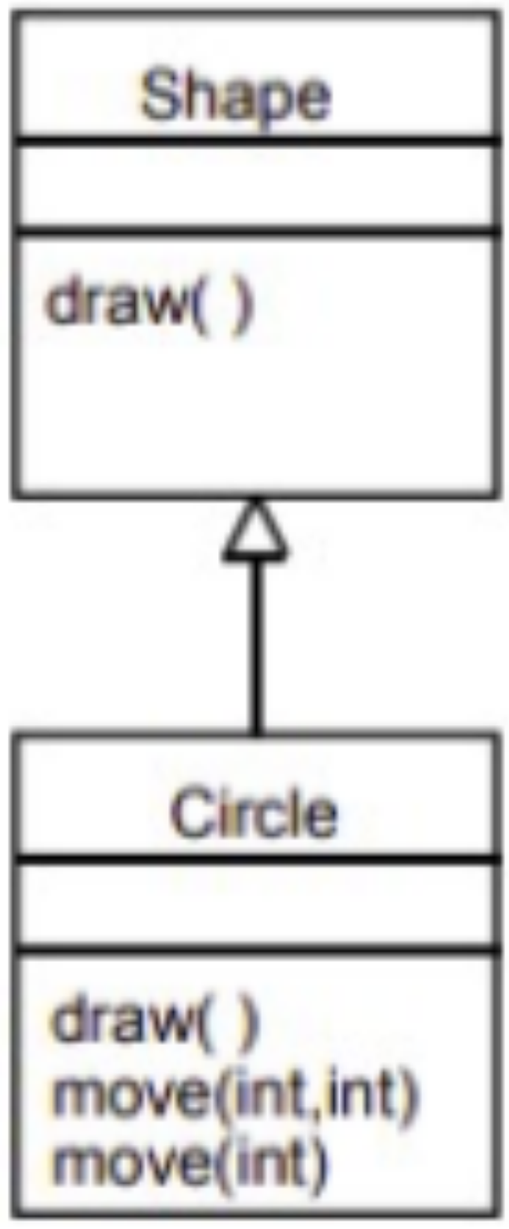
- May not be constructed
- May have variables, methods
- Concrete subclass should override those interfaces.

- What does a Shape look like?
- What code (i.e., algorithm) goes in the Shape draw method?
  - Abstract methods and functions do not need bodies



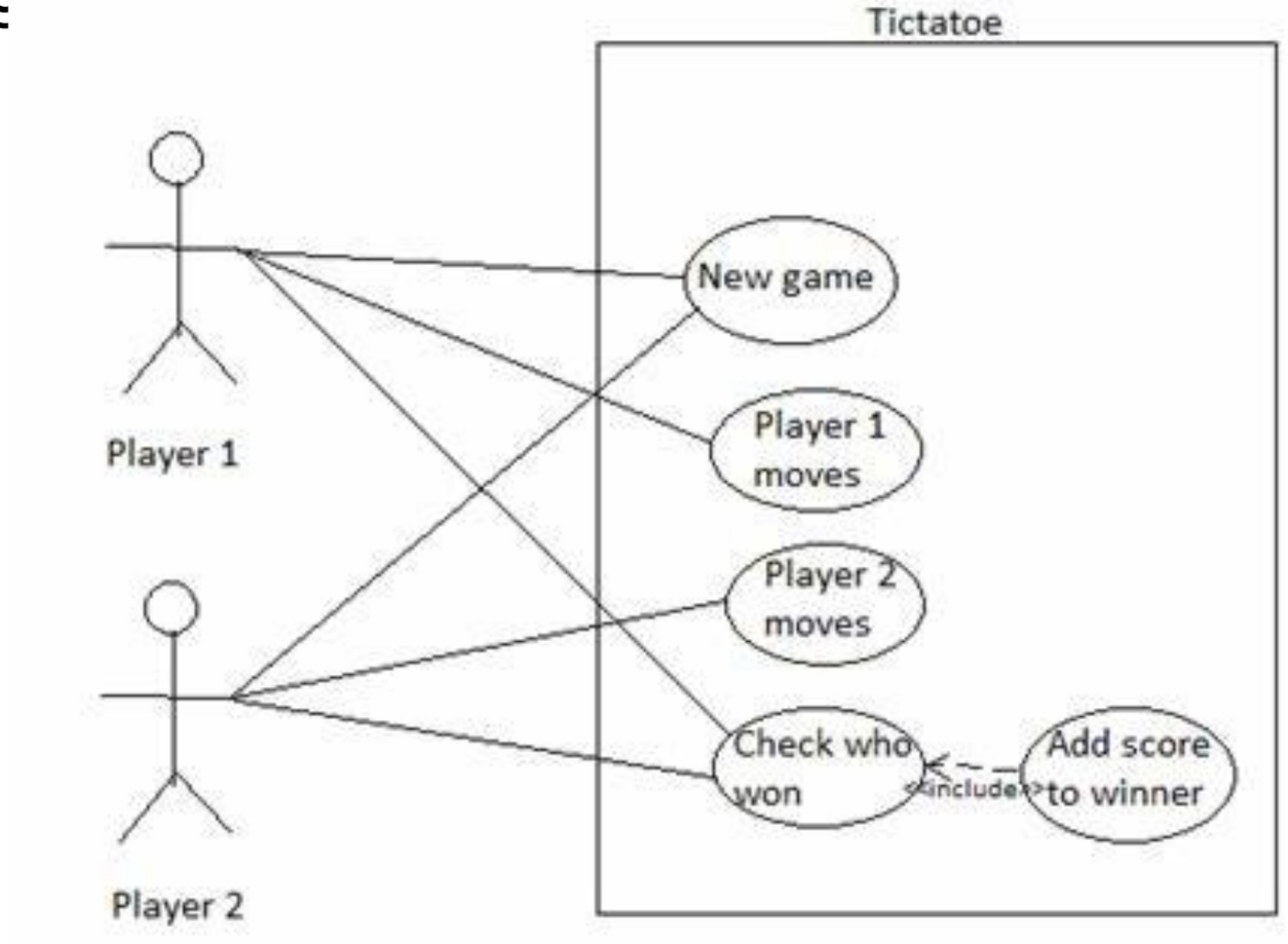
Overriding vs. Overloading

- Overloading
- one class
- two or more methods with the same name but different types of arguments
- Overriding
- The implementation in the subclass overrides the one in the base class with the same name and arguments

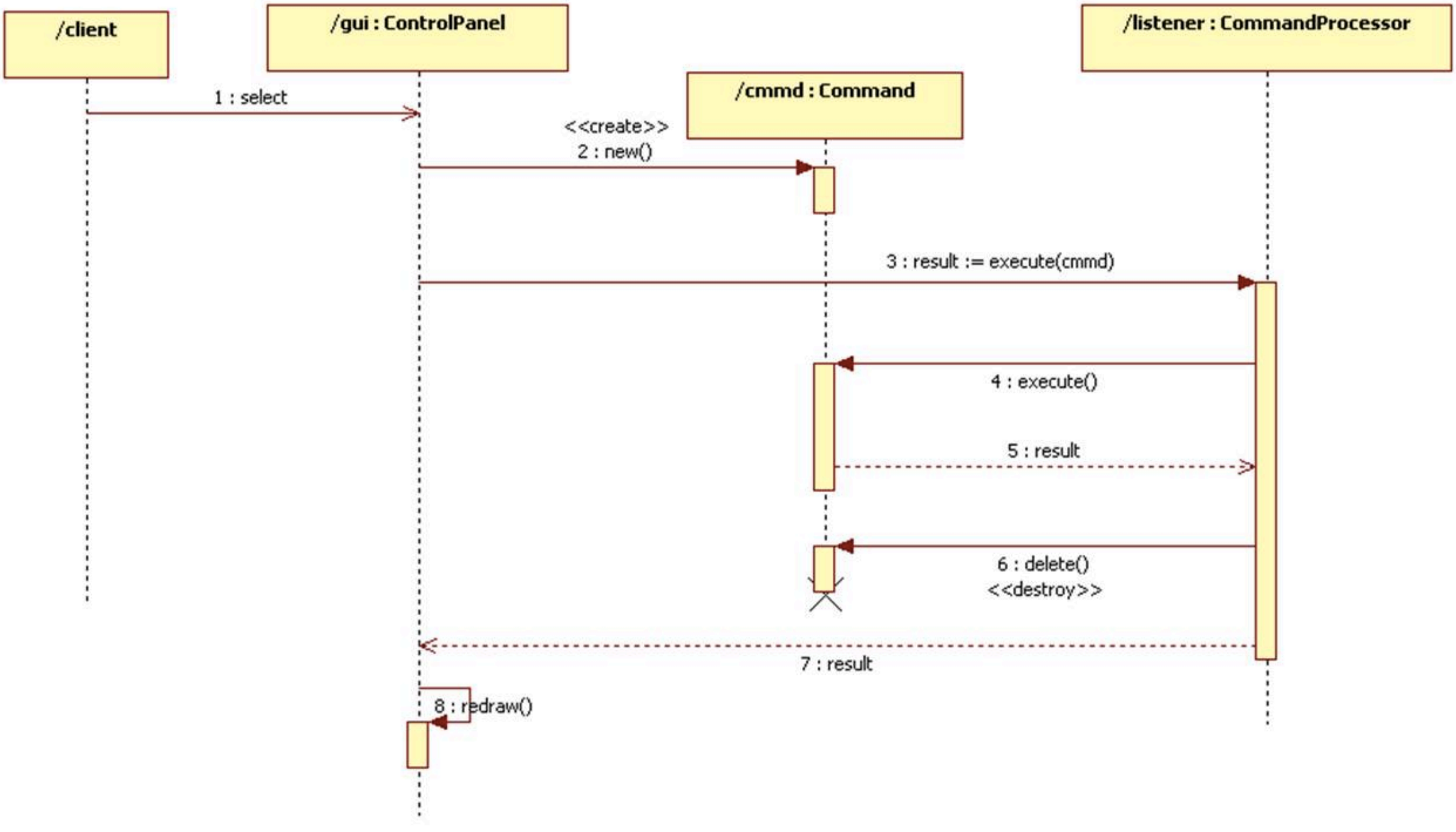


Use Case Diagram

- Models a system at a very high level
- starting point for
- designing class diagram
- interaction diagram
- Readable by a wide range of stakeholders
- No single use case needs to capture everything about a system



Sequence Diagram



End