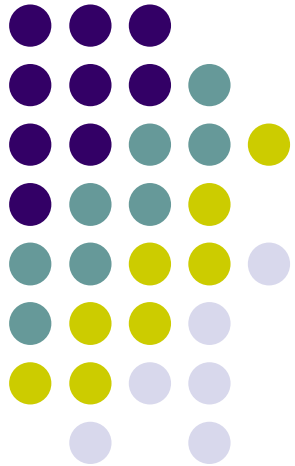
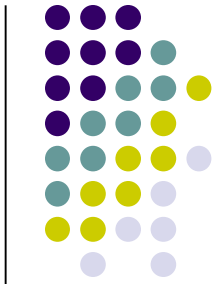
OOP Review



Object-Oriented Programming Revisited

- Key OOP Concepts
  - Object, Class
  - Instantiation, Constructors
  - Encapsulation
  - Inheritance and Subclasses
  - Abstraction
  - Reuse
  - Polymorphism, Dynamic Binding
- Object-Oriented Design and Modeling

Object

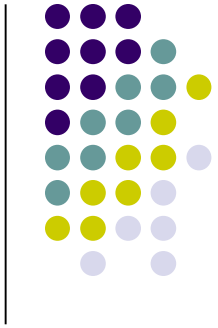


Definition: a thing that has identity, state, and behavior

- identity: a distinguished instance of a **class**
- state: collection of values for its **variables**
- behavior: capability to execute **methods**

* variables and methods are defined in a class

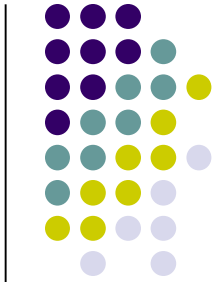
Class



Definition: a collection of data (fields/ variables) and methods that operate on that data

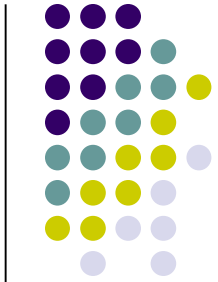
- define the contents/capabilities of the instances (objects) of the class
- a class can be viewed as a *factory* for objects
- a class defines a *recipe* for its objects

Instantiation



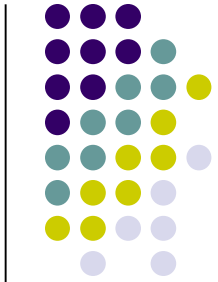
- Object creation
- Memory is allocated for the object's fields as defined in the class
- Initialization is specified through a ***constructor***
  - a special method invoked when objects are created

Encapsulation



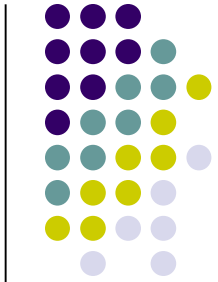
- A key OO concept: “Information Hiding”
- Key points
  - The user of an object should have access only to those methods (or data) that are essential
  - Unnecessary implementation details should be hidden from the user
  - In Java/C++, use classes and access modifiers (public, private, protected)

Inheritance



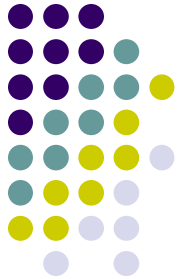
- Inheritance:
  - programming language feature that allows for the implicit definition of variables/methods for a class through an existing class
- Subclass relationship
  - B is a subclass of A
  - B inherits all definitions (variables/methods) in A

Abstraction



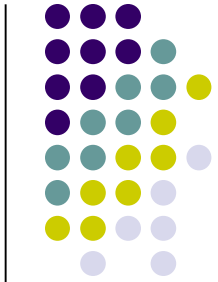
- OOP is about ***abstraction***
- Encapsulation and Inheritance are examples of abstraction
  - What does the verb “abstract” mean?

Reuse



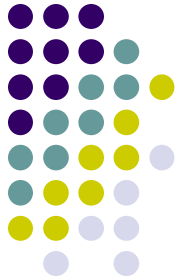
- Inheritance encourages software reuse
- Existing code need not be rewritten
- Successful reuse occurs only through careful planning and design
  - when defining classes, anticipate future modifications and extensions

Polymorphism



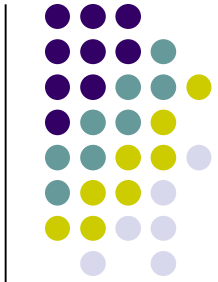
- “Many forms”
  - allow several definitions under a single method name
- Example:
  - “move” means something for a person object but means something else for a car object
- Dynamic binding:
  - capability of an implementation to distinguish between the different forms during run-time

Building Complex Systems



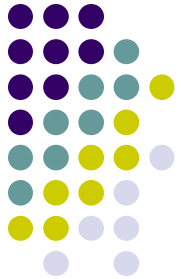
- From Software Engineering:
  complex systems are difficult to manage
- Proper use of OOP aids in managing this complexity
- The analysis and design of OO systems require corresponding modeling techniques

Object-Oriented Modeling



- UML: Unified Modeling Language
  - OO Modeling Standard
  - Booch, Jacobson, Rumbaugh
- What is depicted?
  - Class details and static relationships
  - System functionality
  - Object interaction
  - State transition within an object

Some UML Modeling Techniques



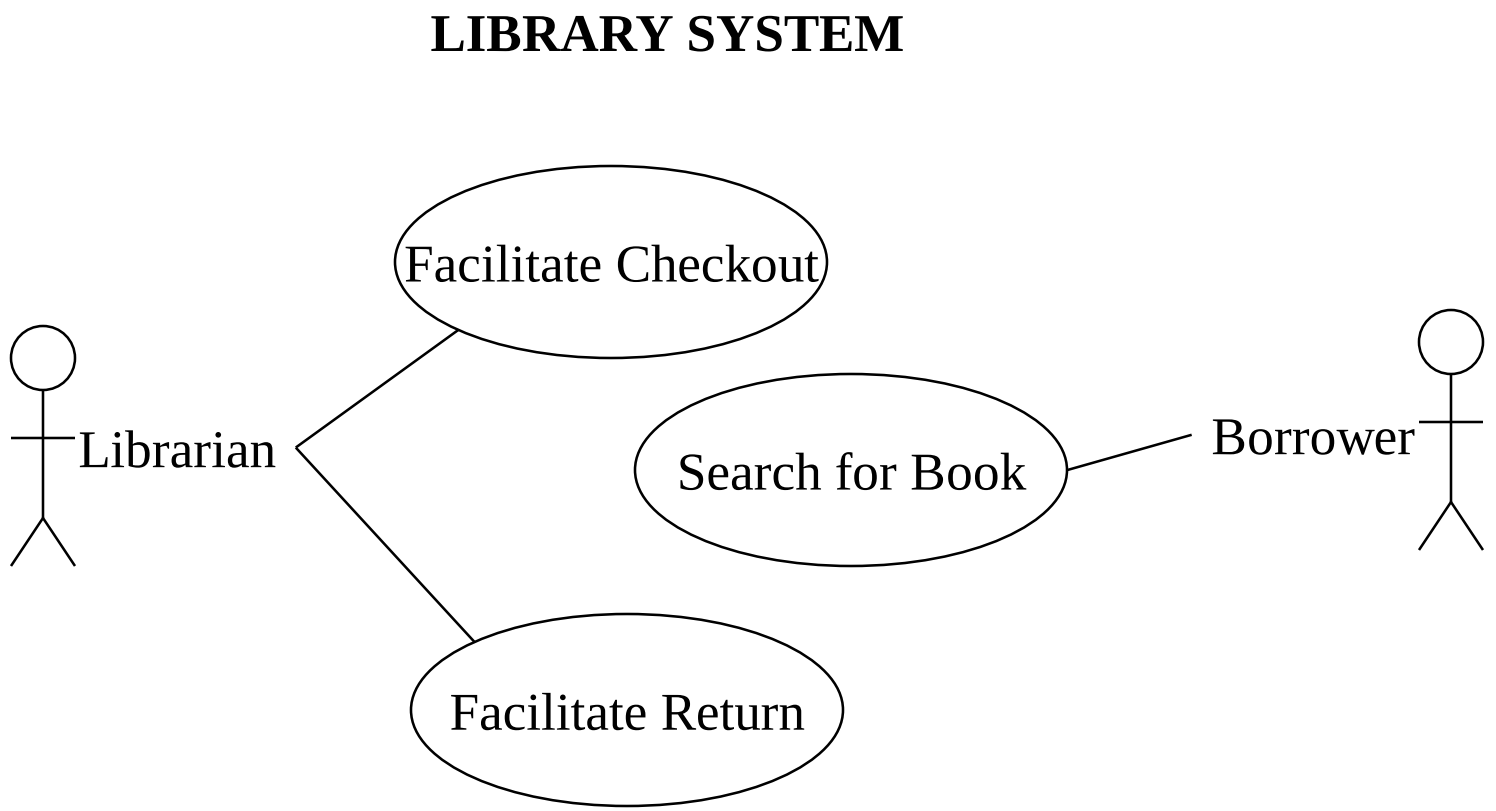
- Class Diagrams
- Use Cases/Use Case Diagrams
- Interaction Diagrams
- State Diagrams

Example: Class Diagram

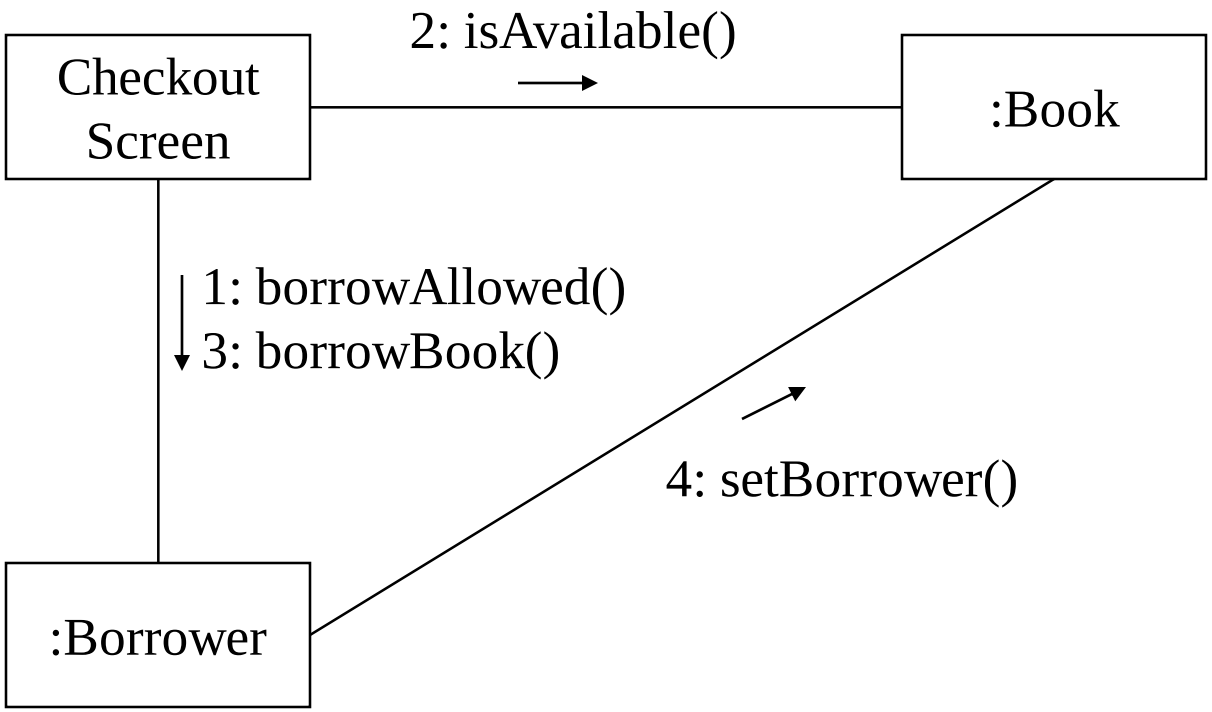

```
public class Borrower {
   Book bk[];
   …
   public Borrower() {
      bk = new Book[3];
   }
}
```

```
public class Book {
   Borrower currBorr;
   …
}
```

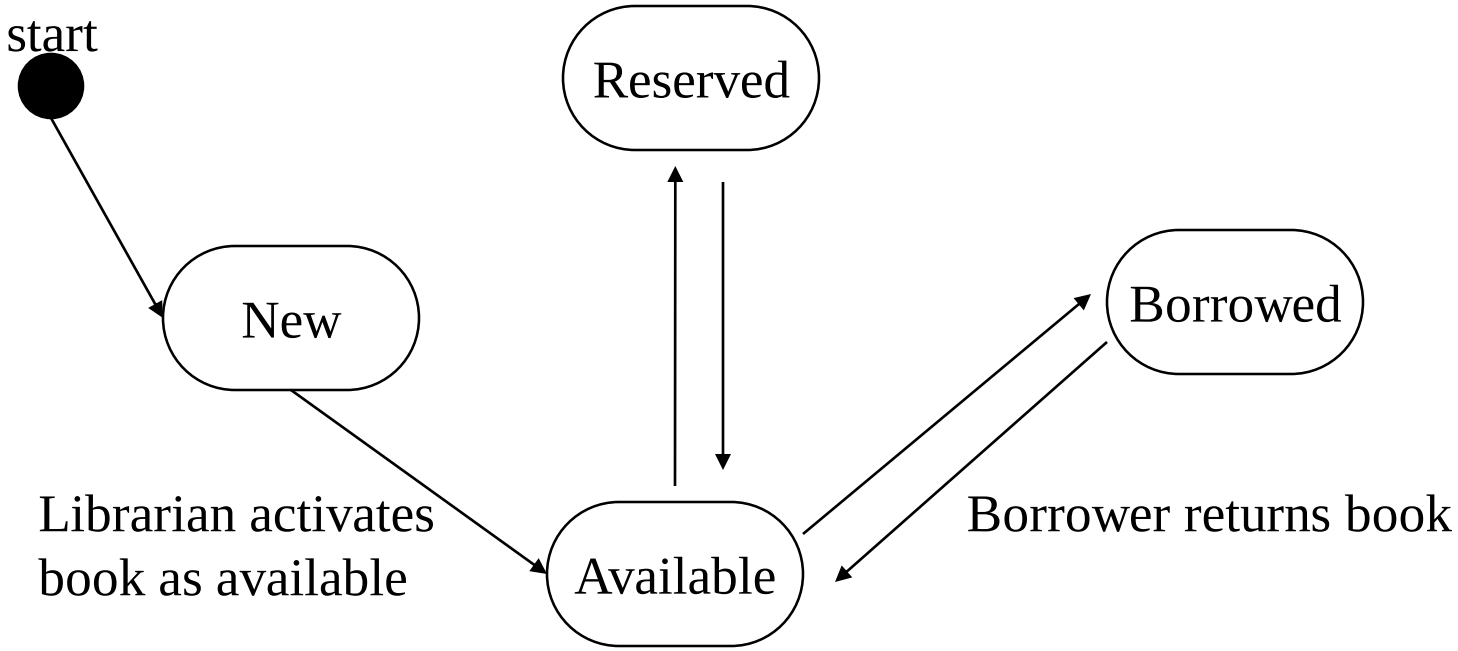
Example:
Use Case Diagram



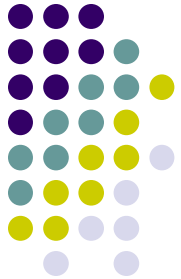
Example: Interaction Diagram



Example:
State Diagram (Book)

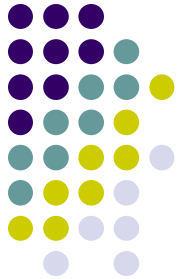


Object-Oriented Design Models



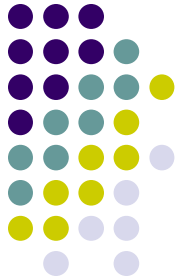
- Static Model
  - Class Diagrams
- Dynamic Model
  - Use Cases, Interaction Diagrams, State Diagrams, others

OO Static Model



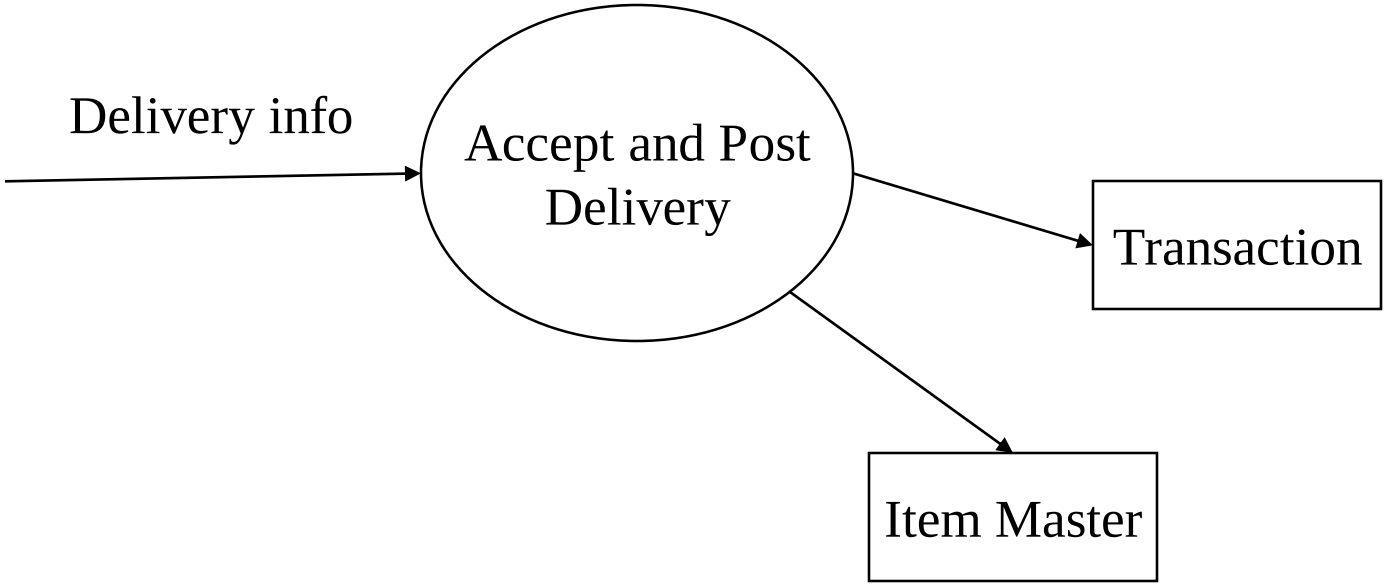
- Classes and Class Diagrams
- Relationships
  - Association
  - Aggregation/Composition
  - Inheritance
- Dependencies
- Attribute and Method names

OO Dynamic Model

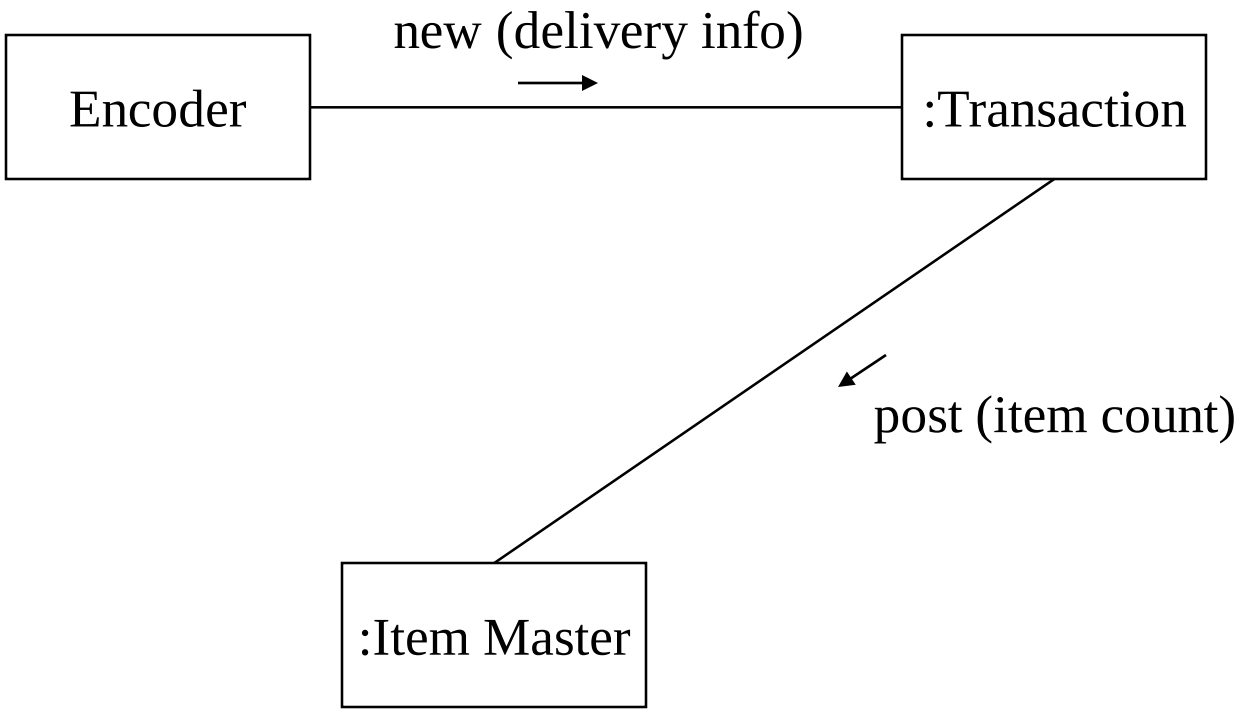


- Goal: Represent
  - Object behavior
  - Object interaction
- Traditional/Procedural Dynamic Modeling
  - Data Flow Diagrams (DFDs)
  - Problem: Processes separate from data
  - Need modeling notation that highlight tight relationship between data & processes

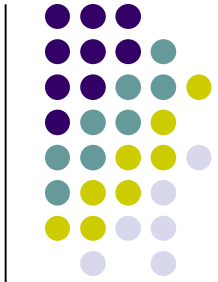
DFD Example (Inventory Management)



OO Counterpart: Object Interaction

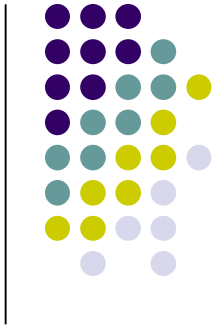


Building an OO Dynamic Model



- Identify use cases
- Describe each use case through an interaction diagram
- For more complex objects, provide a state diagram per class
- Derive implied methods (and attributes)

What's Next?



- Need to understand the notation
- Make sure it helps the software development process
- When to use the UML techniques
  - Primarily when specifying OO design
  - Formal means of communication across the different software development stages