

# Object Oriented Programming and Design in Java

Session 3  
Instructor: Bert Huang

# Announcements

- Next Monday's class canceled for Distinguished Lecture: Feb 1, 11 AM Davis Auditorium.
- Course survey due
- Homework 1 will be posted soon, "officially" out Feb. 3rd

# Review

- basic syntax, javadoc, primitive types, references, importing packages, exceptions, input, Arrays, ArrayLists, declaration keywords, code style
- CUNIX and Eclipse demo

# Today's Plan

- Turning ideas into a program
  - Use cases
  - identifying classes
  - UML diagrams: class diagram, sequence diagram, state diagram
- Example: todo list manager

# Ideas to Programs

Analysis

(common sense)



Design

(object-oriented)



Implementation

(actual programming)

# Phase 1: Analysis

- Ideas or description of final product may be inadequate
- Specifically describe requirements to be considered a completed program
- Decide on exact functionality
- Limit ambition, but don't think too much about design and implementation

# Use Cases

- Use cases specifically describe the operation of the program
- Narrows down exactly what you want your program to do
- Useful as test cases
- Implementation and design don't matter

# Phase 2: Design

- More explicit about object interactions
- Define classes of objects
- Decide responsibilities of classes
- Define attributes and methods of classes



# Identifying Classes

- Good first step: look for **tangible nouns** in use cases. Then...
- **Agents** - objects that perform tasks
- **Events** - store information about events
- **Systems, interfaces** - run the program, talk to user or other programs
- **Foundational classes** - String, Date, etc.

# Identifying Responsibilities

- Good first step: look for verbs, actions in use cases
- These actions may directly describe responsibilities, or
- may depend on other responsibilities

# CRC “Cards”

- Class - Responsibility - Collaborators
- Brainstorming tool for setting up classes and responsibilities
- Collaborators loosely define class relationships; we get more precise later

ClassName	
responsibility 1	Collaborator 1
responsibility 2	Collaborator 2
...	...

# Walkthroughs with CRC

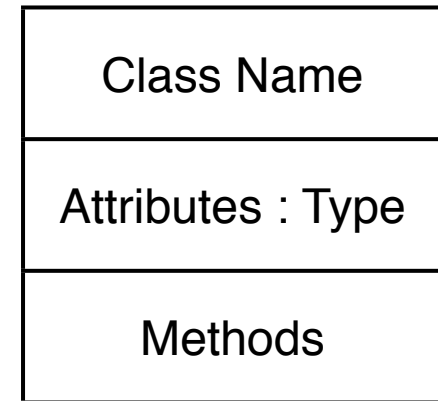
- Play out (partial) use cases using CRC
- Who does what during the use case?
- Do some objects have too much responsibility?
  - Create helper objects or agents
- Are some classes never used?

# Universal Modeling Language

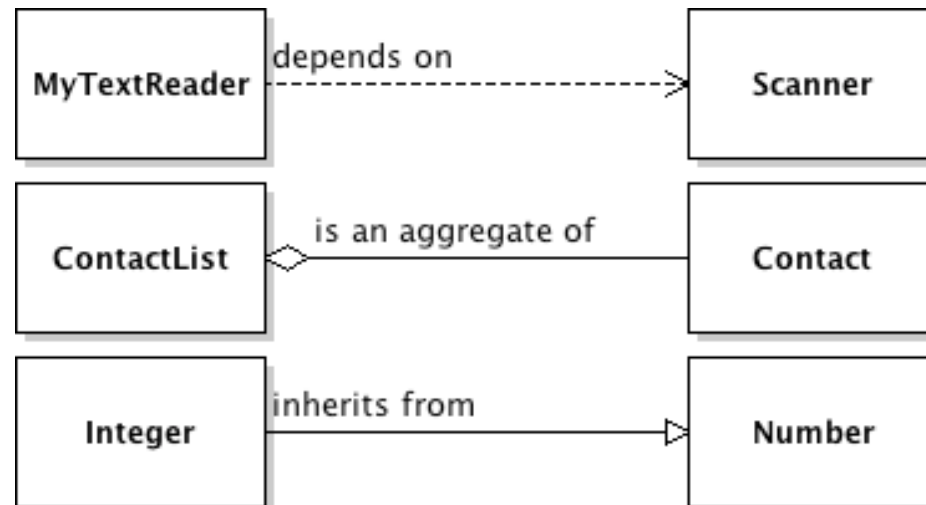
- Standard formatting rules and syntax for modeling software
- More precise than CRC, but still looser than javadoc or actual code skeleton
- Start to name methods based on established responsibilities

# UML Class Diagrams

- Each class is a rectangle



- Connect classes by their relationship

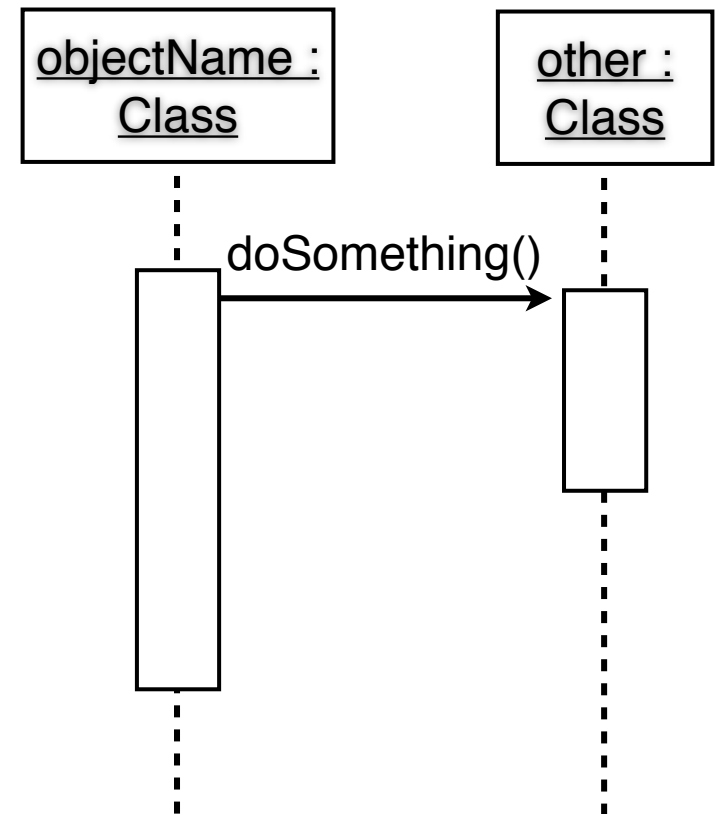


# Class Relationships

- **Dependency** - any time one class needs the other
- **Aggregation** - one class contains elements of the other class
- **Association** - other relationship
- **Inheritance**
- **Interface Implementation**

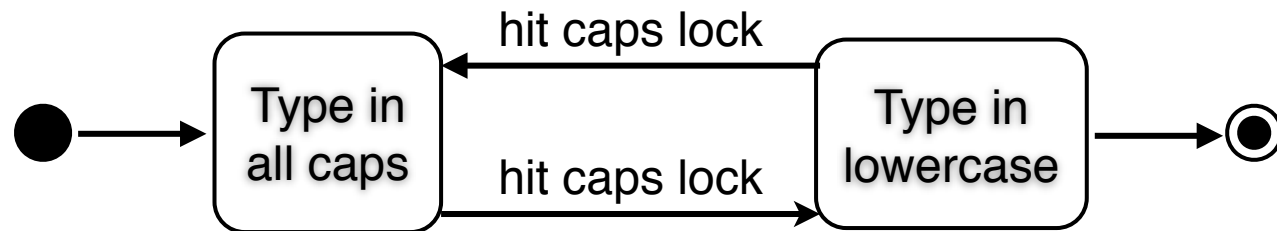
# Sequence Diagrams

- Draw objects as they interact over time
- UML: underline to indicate instances
- Each object has dotted life-line
- **Activation bars** indicate object running
- Arrows indicate method calls





# State Diagrams



- Useful for visualizing how an object changes over time
- Rounded rectangles represent states
- Arrows and text describe triggers for state changes

# Checkpoint

- You should have a tractable design
- Manageable class complexity
- Clean encapsulation
- You can write the code skeleton and javadoc now
- Then Phase 3: Implement

# Example: Console Todo List Manager

- Most programs start with a vague idea:
- Hey, <your name>, make me a program that like helps keep track of stuff I have to do. Or whatever. And it should sort by due date.

# Use Case 1

- User starts the program
- Display saved items numbered and sorted by due date.
- User enters “add laundry” at prompt
- User is prompted for a due date
- User enters date
- list updated and displayed with laundry in its correct sorted position

# Use Case 2

- User has previously entered todo items, including “laundry”
- User starts program
- To do list is displayed
- User enters “finished laundry”
- Laundry is removed from the list, remaining items displayed

# Classes and Responsibilities

- Nouns: date, item, prompt, list
  - Verbs: display, enters, add, delete, update, sort
- Agents: file manager (saving + loading)
- Our classes: TodoItem, TodoPrompt, TodoList, TodoFileManager

# CRC

TodoItem	
Store name, date	TodoList

TodoPrompt	
Display list	TodoList
get commands	

TodoList	
Store list of items	TodoItem
Sort items	TodoPrompt
Add and remove	TodoFileManager

TodoFileManager	
Load list from file	TodoList
Save list to file	

# CRC Walkthrough

TodoItem	
Store name, date	TodoList

TodoPrompt	
Display list	TodoList
get commands	
add/delete TodoItem	

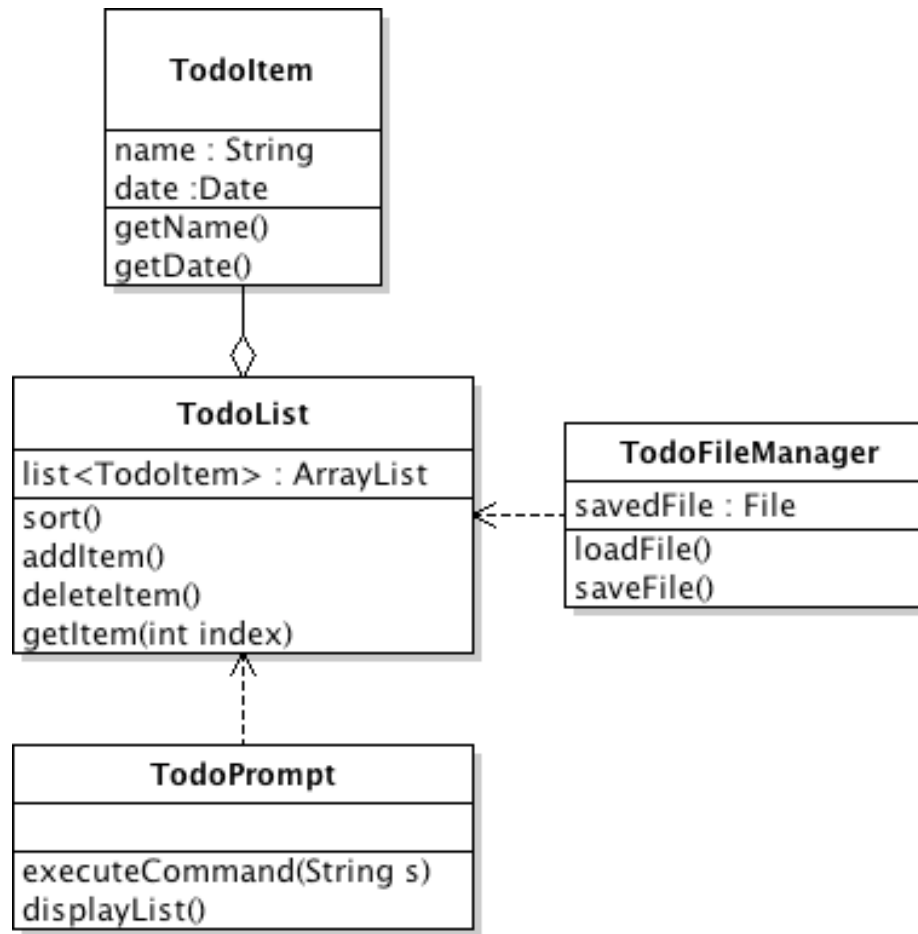
TodoList	
Store list of items	TodoItem
Sort items	TodoPrompt
Add and remove	TodoFileManager

TodoFileManager	
Load list from file	TodoList
Save list to file	

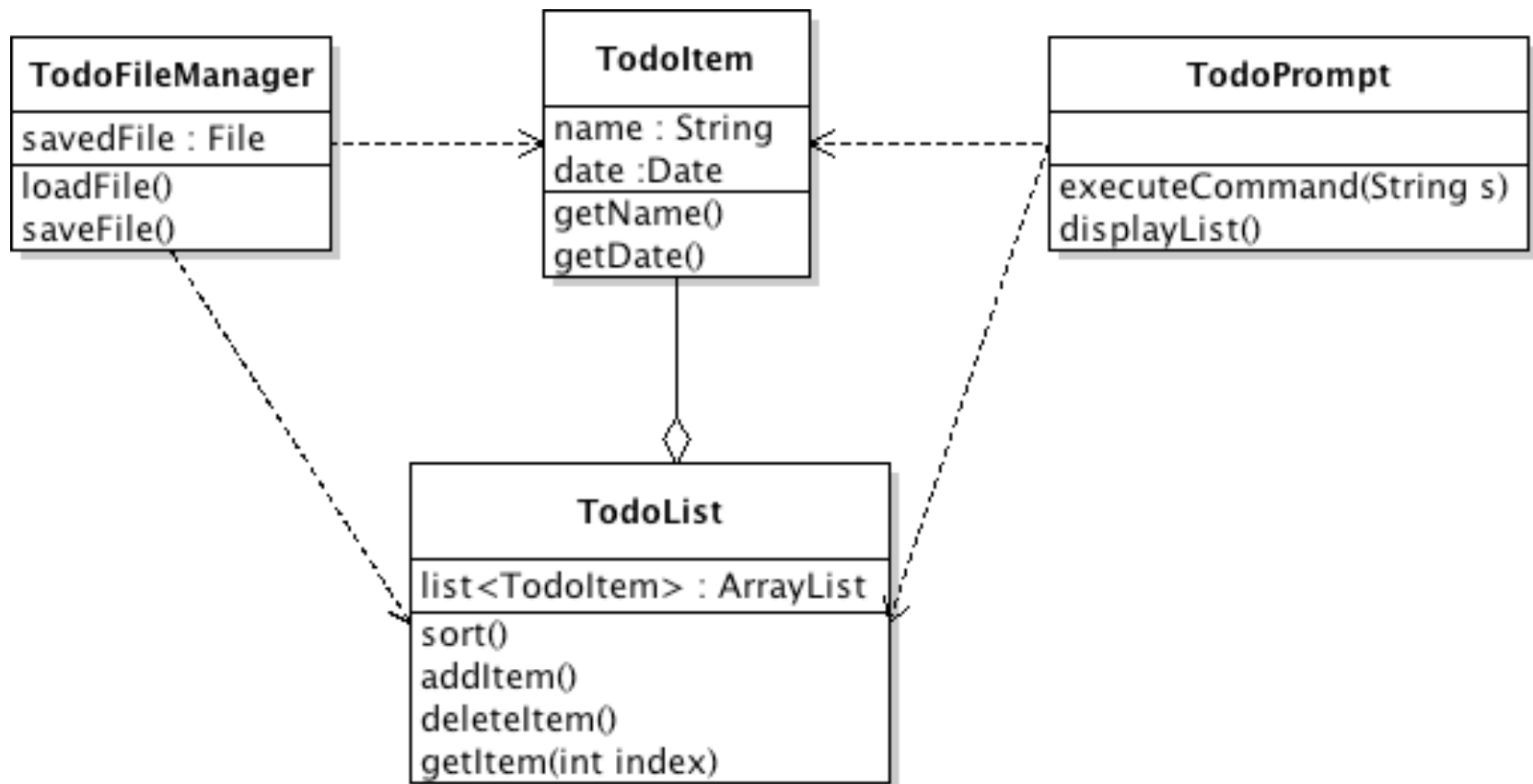
- User starts the program
- Display saved items numbered and sorted by due date.
- User enters “add laundry” at prompt
- User is prompted for a due date
- User enters date
- list updated and displayed with laundry in its correct sorted position



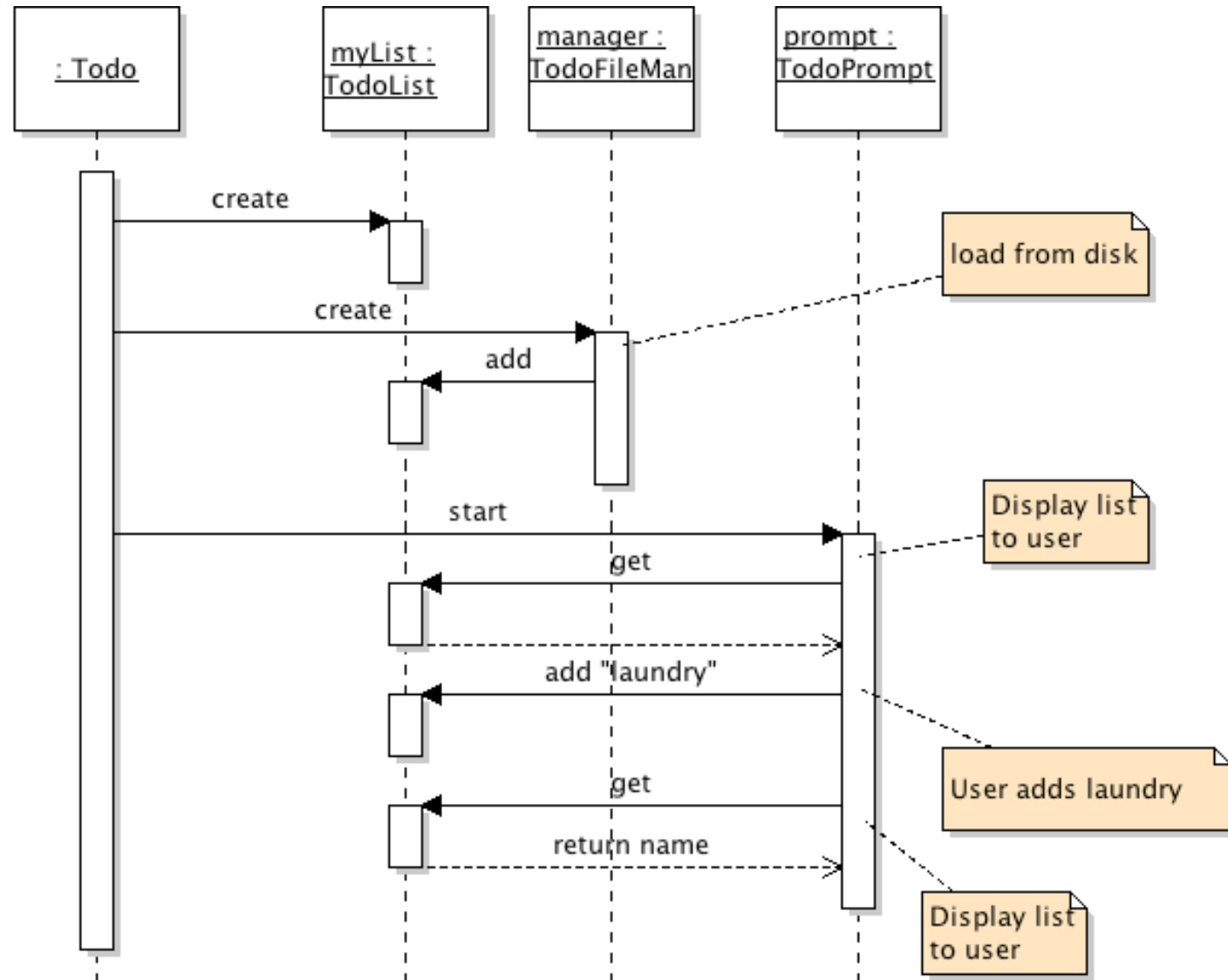
# Class Diagram



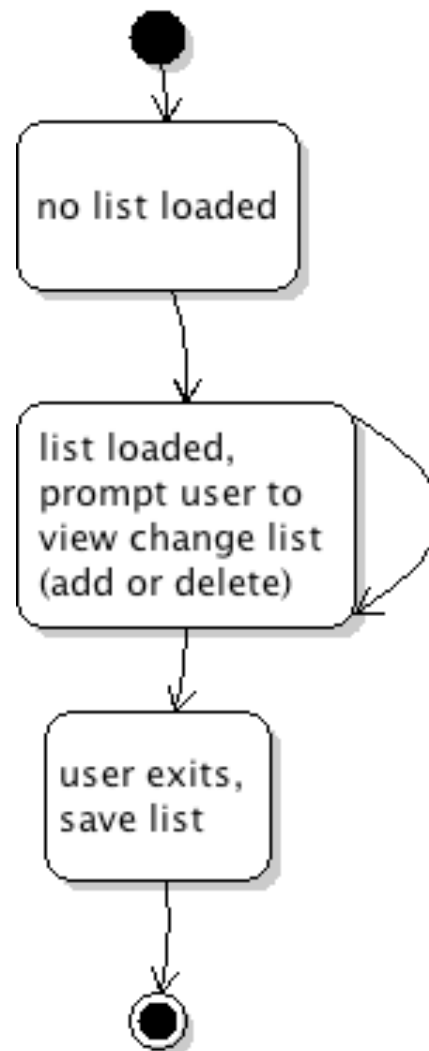
# Class Diagram



# Sequence Diagram 1



# State Diagram



# Violet

- I used Horstmann's Violet to draw the UML diagrams on last few slides
- <http://horstmann.com/violet>

# Reading

- Horstmann Ch. 2
  - Look at his VoiceMail example