Spec Writing

ENGR 297: Spring 2018

Instructor: Andrew Hu

Administrivia

- Welcome to the software-subteam!
- Attendance Link: tinyurl.com/uwigem/18sp/attendance/
- Anonymous Feedback Form: tinyurl.com/uwigem/18sp/feedback/

Group Think Exercise

- https://homes.cs.washington.edu/~mernst/pubs/groupthink-2006-2up.pdf

What are we doing this Spring?

- Learning the fundamentals of software development
- Planning our software as the project comes together from Wetlab
- Writing *some* software

Class vs Team

- Most classes we will act as a class
- As time goes on we will have to act more like a team
- Just keep in mind that we will have to work together differently based on the situation

What are we talking about today

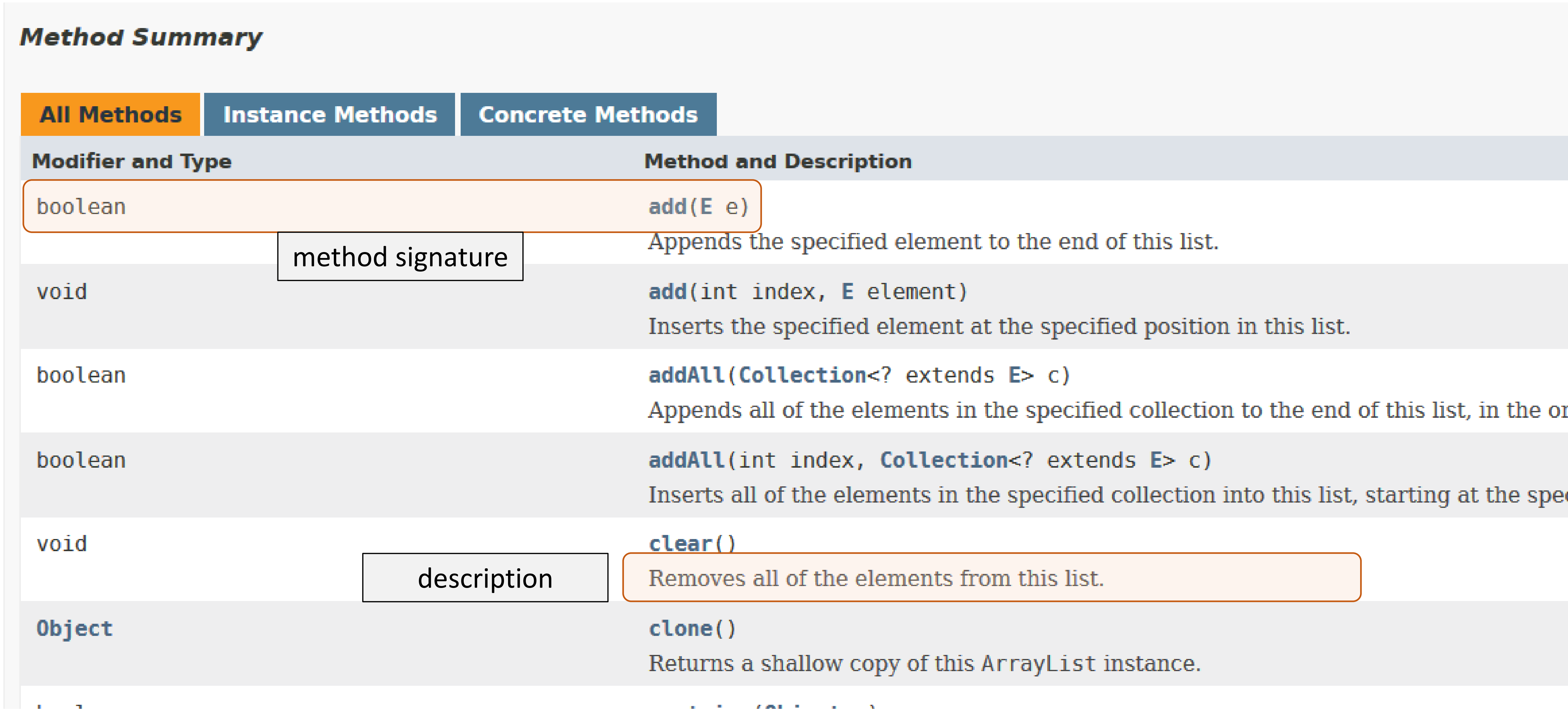
- Specifications
- Why are we writing specs?
- Activity: Let’s try to write a spec

What is a spec? (specification)

collection of class and method signatures

explanations of how to use that library

Example: Java ArrayList



Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
boolean	add(E e) Appends the specified element to the end of this list.
void	add(int index, E element) Inserts the specified element at the specified position in this list.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the or
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the spe
void	clear() Removes all of the elements from this list.
Object	clone() Returns a shallow copy of this ArrayList instance.

method signature

description

How is a spec useful?

formal way to describe a library’s functionality

implementation can be changed without changing interface

hides implementation details from users

Why write the spec before implementing?

think through how the entire system will be used

let others write code that uses your spec, before your code is ready

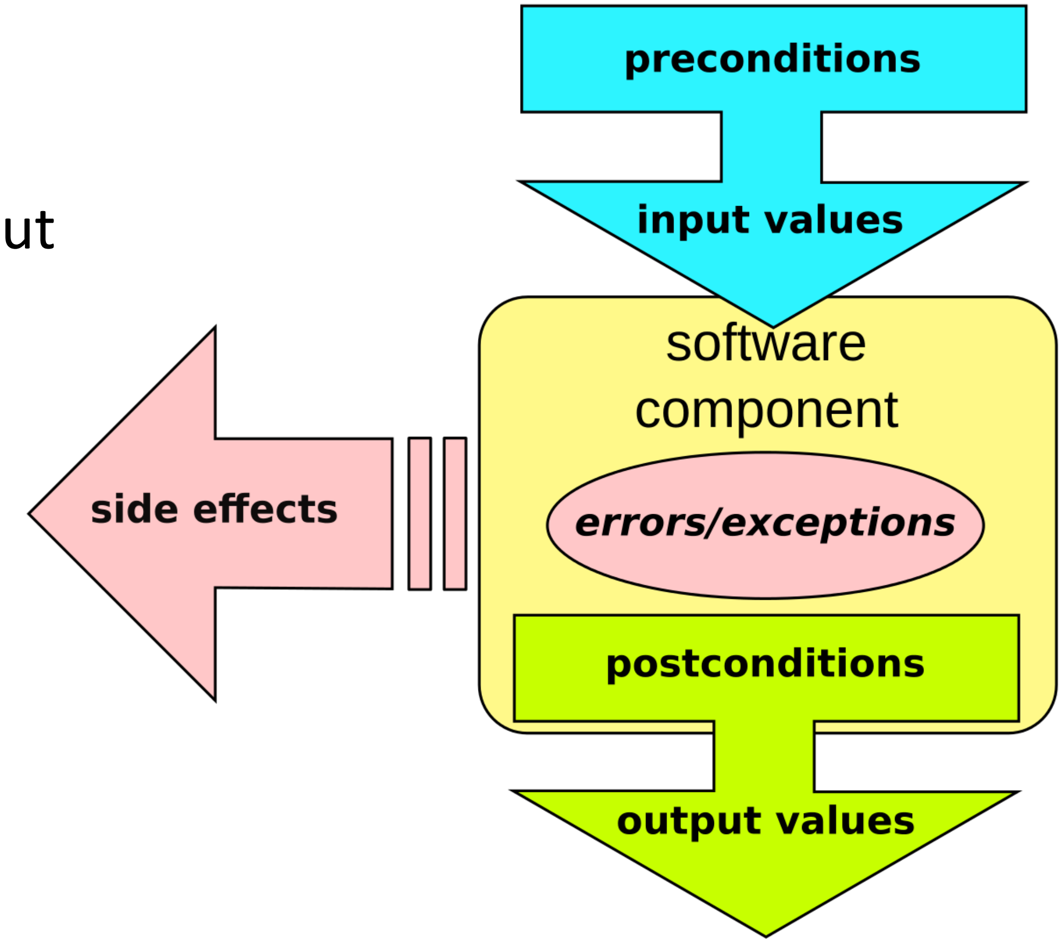
Interface vs Implementation

- Interface
  - How to use this library
  - Just the method calls
  - Black Box

- Implementation
  - The actual code
  - Algorithms and data structures used are visible
  - The inner workings

Precondition & Postcondition

- Preconditions
  - What the user must guarantee about input
- Postconditions
  - What the author must guarantee about output



Pre & Postconditions Examples

- String to int converter

```
static int parseInt(String s){…}
```

- Preconditions?

“s” must be a valid string representation of an integer

- Postconditions?

the value returned is the int value represented by “s”

Undefined Behavior

- Do I have to define what the behavior is when the precondition is met?
  - Yes!
- Do I have to define what the behavior is when the precondition <u>is not</u> met?
  - No, please don't…

Why should we hide the implementation?

we can edit the implementation without having to change the interface

avoid user dependency on information outside of the spec

The Medusa Effect

- Alice’s code uses a large sorted list of data
- Alice tells Bob how she has implemented this
- Bob now assumes that getting the list of data in sorted order is a fast operation, and uses it frequently
- Alice later decides that it is better to implement this as a hash table for better lookup times, while keeping the interface the same
- Now Bob’s code is slowing down the whole system, but he has written so much of it that he can’t just delete it all
- His code has been “turned to stone”

Spec Example: Pacman

Dividing up the parts

First Part: Parts of the Game

How can we divide up Pacman, and what interacts with what?

What are the parts to Pacman?

Pacman

ghosts

map and dots

graphics

Pacman

- What happens when you push the stick/arrow keys in a direction?
- Does he continue to move in that direction?
- What happens when he hits a wall?
- What happens when he eats a big dot?

Ghosts

- What happens when they are released?
- What direction do they move?
- What happens when Pacman eats a big dot?
- What happens when they are eaten by Pacman?
- When do they change directions?

Map & Dots

- What happens when Pacman touches the little dots?
- What happens when Pacman hits a wall?
- What happens when Pacman eats a big dot?

Graphics

- What things can you see on the screen?
- What do you see when Pacman eats a dot?
- What do you see when Pacman eats a big dot?
- What do you see when Pacman eats a ghost?

Any other parts to Pacman?

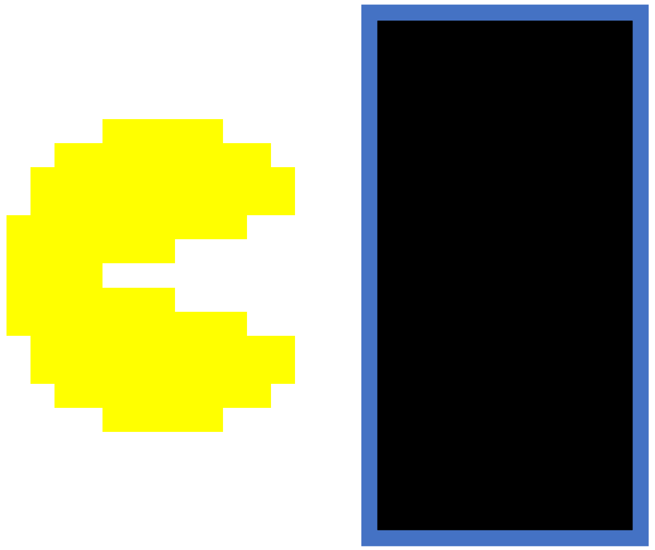
Next Step: Specification

How would these interactions work in the code?

Example: Pacman and Walls

- How do we stop Pacman from running through a wall?
  - Check for collision before updating position
- Who should check? Pacman or the map?
- Who has the "right" to update Pacman's coordinates?
- No on true answer

Give an argument for both sides: Pacman or the map updating coordinates



Activity: Brainstorming

- Try to come up with an outline for a Pacman spec
- Think about which part controls what
  - Does the map or Pacman control his coordinates?
  - Do Pacman and the ghosts write directly to the screen? Does it go through the map?
  - When updating Pacman’s position, do the dots need to be updated? Is that a separate method? If so, who owns it?