# CSE 331
# Software Design & Implementation

Winter 2026

Section 3 – ADTs

# Administrivia

- HW3 released tonight - **Due next Wed 11:59pm**

# Specifications for ADTs – Review

- New Terminology for specifying ADTs:
  - **Abstract State / Representation (Math)**
    - How clients should understand the object
    - Ex: `List(nil or cons)`
  - **Concrete State / Representation (Code)**
    - Actual fields of the record and the data stored
    - Ex:

```
public class List {
    final int hd;
    final List tl;
```

# State Representations

- We've had different abstract and concrete types all along!
  - in our math, List is an inductive type (abstract)
  - in our code, `List` is a class with two fields (concrete)

- Term "object" (or "obj") will refer to abstract state
  - "object" means mathematical object (representative of the class as a whole or as an idea)
  - "obj" is the mathematical value that the record represents (similar to a specific instance of a class)

# Internally Documenting ADTs – Review

**Abstract Function (AF)** – defines what abstract state the field values represent

- Maps field values → the object they represent
- Output is math, this is a mathematical function

**Representation Invariants (RI)** – facts about the field values that must always be true

- Constructor must always make sure RI is true at runtime
- Can assume RI is true when reasoning about methods
- AF only needs to make sense when RI holds
- Must ensure that RI *always* holds

# Documenting ADTs – Example

```
// A list of integers that can retrieve the last element in O(1)
interface FastList {
  /**
  * Returns the object as a regular list
  * @returns obj
  */
  List toList();
}

class FastLastList implements FastList {
  // RI: this.last = last(this.list);
  // AF: obj = this.list;

  // @returns last(obj)
  int getLast() {
    return this.last;
  };
}
```

Talk about functions in terms of the abstract state (obj)

Hide the representation details (i.e. real fields) from the client