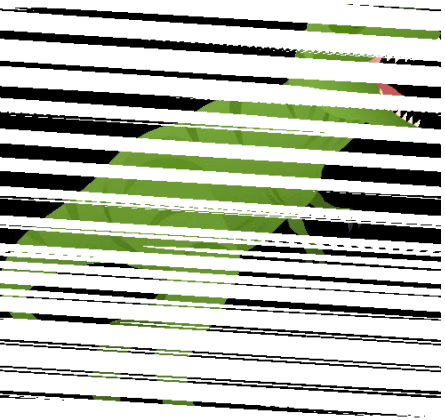
CSE 331
Software Design & Implementation

Winter 2026

Section 1 – Specifications

Welcome!

- Let’s all introduce ourselves:
  - Name and pronouns
  - Year
  - What other classes you are taking this quarter
  - Would you rather be a dinosaur or a unicorn and why?



Homework Cycle

- Section:
  - Every Thursday
  - Content review & practice problems
  - Graded on participation
  - (If sick): Can submit on gradescope for participation credit
- Homework:
  - Released Thursday evenings
  - Due on Gradescope **@ 11:59pm Wednesday**
  - TAs will assess and give feedback
  - Graded on **correctness**

Review - Abstraction

- Hides unnecessary details from clients via **specification**
- Creates an *Abstraction Barrier* between clients and implementers
  - Implementers promise that the code follows the specification
  - Clients promise not to rely on details outside the specification

Poll - Abstraction

What is one downside of AI regarding abstraction?

AI does not respect abstraction barriers. It relies on implementation details when writing code rather than solely specifications.

This leads to dependent code, where a change in implementation of a function can break the rest of the code.

Review – Specification Types

- **Imperative** specification says how to calculate the answer
  - Gives the exact steps to get the answer
  - Just have to translate math to code
  - Ex: Absolute value: $|x| = x$ if $x \geq 0$ and $-x$ otherwise

- **Declarative** specification says what the answer looks like
  - Does not say how to calculate it
  - Up to us to ensure that our code satisfies the spec
  - Ex: Subtraction (a – b): return x such that b + x = a

Question - Specification Types

What type is each of these specs?:

- Steps to calculate the square of a number n: multiply n by itself, n * n.
  - Declarative
  - Imperative

- Return a number, such that this number is the same value as the square of n
  - Declarative
  - Imperative

Question - Specification Types

What type is each of these specs?:

- Steps to calculate the square of a number n: multiply n by itself, n * n.
  - Declarative
  - **Imperative**

- Return a number, such that this number is the same value as the square of n
  - Declarative
  - Imperative

Question - Specification Types

What type is each of these specs?:

- Steps to calculate the square of a number n: multiply n by itself, n * n.
  - Declarative
  - **Imperative**

- Return a number, such that this number is the same value as the square of n
  - **Declarative**
  - Imperative

Review – Specifications

A specification consists of two parts:

Precondition: Allowed inputs

Postcondition: Allowed outputs

A specification is **stronger** when it has *less restrictive inputs* and *more restrictive outputs*. In other words, strong specifications have more guarantees for a larger set of inputs.

Question - Specifications

Which of the two specs is stronger?:

```
/** Returns the square root of a given number x
 * @requires x >= 0
 * @return the integer y such that y^2 = x
```

Or

```
/** Returns the square root of a given number x
 * @requires x >= 0
 * @return the integer y such that y^2 = x and y >= 0
```

Question - Specifications

Which of the two specs is stronger?:

```
/** Returns the square root of a given number x
 * @requires x >= 0
 * @return the integer y such that y^2 = x
```

Or

```
/** Returns the square root of a given number x
 * @requires x >= 0
 * @return the integer y such that y^2 = x and y >= 0
```

Review – Math Notation

Standard notations

Made up for this class

$\mathbb{N}$	all non-negative integers ("natural" numbers)
$\mathbb{Z}$	all integers
$\mathbb{R}$	all real numbers
$\mathbb{B}$	the boolean values (T and F)
$\mathbb{S}$	any character
$\mathbb{S}^*$	any sequence of characters ("strings")

- **Union**: $A \cup B$ set including everything in A *and* B
- **Tuple**: $A \times B$ all pairs (a, b) where a $\in A$ and b $\in B$
- **Record**: $\{x: A, y: B\}$ all records with fields x, y of types A, B

Review – Math Notation

- **Side Conditions**: limiting / specifying input in right column
  - ex: $abs : \mathbb{R} \rightarrow \mathbb{R}$

    $abs(x) := x \quad \text{if } x \geq 0$

    $abs(x) := -x \quad \text{if } x < 0$
  - conditions must be **exclusive** and **exhaustive**
- **Pattern Matching**: defining function based on input cases
  - Exactly **one** rule for every valid input

    ex: $f : \mathbb{N} \rightarrow \mathbb{N}$

    $f(0) := 0$

    $f(n+1) := n$
  - "n + 1" signifies that input must be > 0 since smallest $\mathbb{N}$ would be 0
  - Preferred over side conditions in most cases
- Course Website > Topics > Math Notation Notes

Review - Math Notation Example

Consider the following function, which calculates half when given an **even** number but also accepts other inputs (though it doesn't perform the same behavior in those cases):

$$\text{half} : (\text{undefined} \cup \mathbb{N}) \to \mathbb{Z}$$

$$\begin{aligned}
\text{half}(\text{undefined}) &:= 0 \\
\text{half}(n) &:= n/2 && \text{if } n \text{ is even} \\
\text{half}(n) &:= -(n+1)/2 && \text{if } n \text{ is odd}
\end{aligned}$$