Class 6: Abstract Data Types

6.102 — Software Construction
Spring 2024

Sudoku ADT

Exercise: yellkey.com/**send**

Nanoquiz: yellkey.com/**garden**

Start converting `Sudoku` into an abstract data type

- Uncomment the `export class Sudoku { ... }` declaration
- Change `puzzleGrid` to an instance variable
- Change the functions to instance methods
    - `sudokuTest` has already been changed,
      so you're fixing its compilation errors
      without changing `sudokuTest`

Don't change `blockSize` and `puzzleSize` (yet),
they can remain constants

Nanoquiz

- This quiz is just for you and your own brain:
  - closed-book, closed-notes
  - nothing else on your screen
- Lower your laptop screen when you're done

yellkey.com/**garden**

Recipe for an ADT

Recipe for an ADT

(Let’s pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

Recipe for an ADT

(Let’s pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

`puzzleGrid.length = puzzleSize`

Recipe for an ADT

(Let’s pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

`puzzleGrid.length = puzzleSize` ✓

`puzzleGrid[i].length = puzzleSize` for all 0 ≤ `i` < `puzzleSize`

a.k.a. `puzzleGrid` is a square array `puzzleSize` × `puzzleSize`

Recipe for an ADT

(Let’s pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

`puzzleGrid.length = puzzleSize` ✓

`puzzleGrid[i].length = puzzleSize` for all 0 ≤ `i` < `puzzleSize` ✓

a.k.a. `puzzleGrid` is a square array `puzzleSize` × `puzzleSize`

`puzzleGrid[i][j]` integer *s.t.* 1 ≤ `puzzleGrid[i][j]` ≤ `puzzleSize`
for all 0 ≤ `i`, `j` < `puzzleSize`

Recipe for an ADT

(Let’s pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

`puzzleGrid.length = puzzleSize` ✓

`puzzleGrid[i].length = puzzleSize` for all 0 ≤ `i` < `puzzleSize` ✓

a.k.a. `puzzleGrid` is a square array `puzzleSize` × `puzzleSize`

`puzzleGrid[i][j]` integer *s.t.* 0 ≤ `puzzleGrid[i][j]` ≤ `puzzleSize`
for all 0 ≤ `i`, `j` < `puzzleSize`

a.k.a. `puzzleGrid[i][j]` in { 0, 1, …, `puzzleSize` } for all 0 ≤ `i`, `j` < `puzzleSize`

Recipe for an ADT

(Let’s pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

`puzzleGrid.length = puzzleSize` ✓

`puzzleGrid[i].length = puzzleSize` for all 0 ≤ `i` < `puzzleSize` ✓

a.k.a. `puzzleGrid` is a square array `puzzleSize` × `puzzleSize`

`puzzleGrid[i][j]` integer *s.t.* 0 ≤ `puzzleGrid[i][j]` ≤ `puzzleSize`
for all 0 ≤ `i`, `j` < `puzzleSize`

a.k.a. `puzzleGrid[i][j]` in { 0, 1, …, `puzzleSize` } for all 0 ≤ `i`, `j` < `puzzleSize`

$\boxed{\text{no number appears more than once in any row, column, or block}}$

Recipe for an ADT

(Let's pause for a second)

What are the preconditions on the `puzzleGrid: number[][]` parameter?

`puzzleGrid.length = puzzleSize` ✓

`puzzleGrid[i].length = puzzleSize` for all 0 ≤ `i` < `puzzleSize` ✓

a.k.a. `puzzleGrid` is a square array `puzzleSize` × `puzzleSize`

`puzzleGrid[i][j]` integer *s.t.* 0 ≤ `puzzleGrid[i][j]` ≤ `puzzleSize`
for all 0 ≤ `i`, `j` < `puzzleSize`

a.k.a. `puzzleGrid[i][j]` in { 0, 1, …, `puzzleSize` } for all 0 ≤ `i`, `j` < `puzzleSize`

no positive number appears more than once in any row, column, or block

Exercise: yellkey.com/**send**

Recipe for an ADT

Finish making Sudoku into an ADT

- Change `puzzleGrid` to an instance variable
- Change the functions to instance methods
- `sudokuTest` cases should all pass
- Update the specs on `Sudoku` methods
- Update the `Sudoku` class overview comment
  - Clients aren’t passing `puzzleGrid: number[][]` to every method, so do they still encounter those preconditions? Where?

And then finally:

- How can we improve `blockSize` and `puzzleSize`?

Testing an ADT

Let’s test `solve : Sudoku → boolean`

Implemented as an instance method `public solve(): boolean { ... }`

Which of these could we partition?

A. initial value of `this`
B. initial value of `puzzleGrid`
C. value of `puzzleSize`
D. solving algorithm
E. return value

Testing an ADT

Let’s test `solve : Sudoku → boolean`

If we want to partition on whether the puzzle (that is, `this`)
is missing 0, 1, or more than 1 number, how do we express that?

number of…

A. *filled-in cells*
B. *blank cells*
C. *zeros*
D. *zeros in* `puzzleGrid`
E. *zeros in subarrays of* `puzzleGrid`

… is: 0, 1, >1

Testing an ADT

Let's test `solve : Sudoku → boolean`

Suppose we use test puzzle `p`, which has exactly one solution.
After calling `p.solve()`, which assertions are reasonable to include?

assert that…

A. calling `solve()` returned true
B. `p.isSolved()` returns true
C. `p.puzzleGrid` matches the solution
D. `p.toString()` returns the solution formatted per the provided code

Choosing reps

We have a creator `number[][] → Sudoku`

Does the creator force the representation to be `number[][]`?

Choosing reps

We have a creator `number[][] → Sudoku`

Does the creator force the representation to be `number[][]`? No!

Invent 4 new reps for `Sudoku`:

- a rep using just one array
- a rep using *no* arrays
- a redundant rep that makes observers easy to write
- a rep that doesn't use special values (like 0) for blanks

Write your new reps as fields in comments at the top of `sudoku.ts`

- include for each rep a comment explaining what it means