Class 7: Abstraction Functions & Rep Invariants

6.102 — Software Construction
Spring 2024

How abstract data types came about

Exercise:

central

Nanoquiz:

pick

How abstract data types came about

Exercise:
central

Nanoquiz:
pick

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/**pick**

Exercise: central

Finishing the Sudoku ADT

In `sudoku.ts`, first work on the rep invariant:

1. Rep invariant: do the TODO
2. `checkRep()`: do the TODO
3. Call `checkRep()` everywhere you need to
4. Get the tests passing

Exercise: central

Finishing the Sudoku ADT

In `sudoku.ts`, first work on the rep invariant:

1. Rep invariant: do the TODO
2. `checkRep()`: do the TODO
3. Call `checkRep()` everywhere you need to
4. Get the tests passing

If you finish that:

5. Write the abstraction function
6. Write the rep exposure safety argument

RI: *probably* complete or incomplete?

RI: *probably* complete or incomplete?

```
export class PlayerStats {
  private totalPointsScored: number;
  private pointsScoredInOvertime: number;
  // AF: ...

  // RI:
  //   pointsScoredInOvertime <= totalPointsScored
}
```

RI: *probably* complete or incomplete?

```
export class PlayerStats {
  private totalPointsScored: number;
  private pointsScoredInOvertime: number;
  // AF: ...

  // RI:
  //   pointsScoredInOvertime <= totalPointsScored
}
```

Probably INCOMPLETE

```
  // RI:
  //   totalPointsScored and pointsScoredInOvertime are nonnegative integers
```

… but what AF could we use instead?

RI: *probably* complete or incomplete?

```
export class Team {
  private people: Array<string>;
  // AF: ...

  // RI:
  //   people.size() >= 2
  //   all strings in people are nonempty
}
```

RI: *probably* complete or incomplete?

```
export class Facebook {
  private users: Set<User>;
  private friends: Map<User, Set<User>>;
  // AF: ...

  // RI:
  //   u1 in friends.get(u2) iff u2 in friends.get(u1)
}
```

Finishing the Sudoku ADT

Exercise: central

In `sudoku.ts`:

1. Rep invariant: do the TODO
2. `checkRep()`: do the TODO
3. Call `checkRep()` everywhere you need to
4. Get the tests passing

5. Write the abstraction function
6. Write the rep exposure safety argument

AF: good or bad?

AF: good or bad?

```
export class Player {
  private name: string;
  private birthday: Date;
  // RI: ...

  // AF: the player's name is stored in name,
  //     and the player's birthday is stored in birthday
}
```

AF: good or bad?

```
export class Complex {
  private parts: number[];
  // RI: ...

  // AF(parts) = the complex number parts[1] + i*parts[0]
}
```

AF: good or bad?

```
export class LineSegment {
  private start, end: Point;
  private length: number;
  // RI: ...

  // AF(start,end,length) = the line segment between `start` and `end`
}
```

AF: good or bad?

```
export class Time {
  private s: number;
  // RI: ...

  // AF(s) is a time of day
}
```

Rep: safe or exposed?

Rep: safe or exposed?

```
export class Team {
  private people: Array<string>;

  public pickSomebody(): string {
    return this.people[0];
  }
  ...
}
```

Rep: safe or exposed?

```
export class Team {
  private people: Array<string>;

  public pickSomebody(): string {
    return this.people[0];
  }
  ...
}
```

Rep: safe or exposed?

```
export class Team {
  private readonly people: ReadonlyArray<string>;

  public constructor(people: ReadonlyArray<string>) {
    this.people = people;
  }

  public getMembers(): ReadonlyArray<string> {
    return this.people;
  }
  ...
}
```

Rep: safe or exposed?

```
export class Team {
  private readonly people: ReadonlyArray<string>;

  public constructor(people: ReadonlyArray<string>) {
    this.people = people;
  }

  public getMembers(): ReadonlyArray<string> {
    return this.people;
  }
  ...
}
```

Rep: safe or exposed?

```
export class Team {
  private readonly people: ReadonlyArray<string>;

  public constructor(people: ReadonlyArray<string>) {
    this.people = people;
  }

  public getMembers(): ReadonlyArray<string> {
    return this.people;
  }
  ...
}
```

Rep: safe or exposed?

```
export class HallOfFame {
  private readonly records: Map<Sudoku, number> = new Map();

  /** @param time >= 0 */
  public addRecord(puzzle: Sudoku, time: number): void {
    const record = this.records.get(puzzle);
    if (record === undefined || time < record) {
      this.records.set(puzzle, time);
  }

  /** @param puzzle must have been added */
  public getRecord(puzzle: Sudoku): number {
    return this.records.get(puzzle) ?? assert.fail();
  ...
}
```

Rep: safe or exposed?

```
export class HallOfFame {
  private readonly records: Map<Sudoku, number> = new Map();

  /** @param time >= 0 */
  public addRecord(puzzle: Sudoku, time: number): void {
    const record = this.records.get(puzzle);
    if (record === undefined || time < record) {
      this.records.set(puzzle, time);
  }

  /** @param puzzle must have been added */
  public getRecord(puzzle: Sudoku): number {
    return this.records.get(puzzle) ?? assert.fail();
  ...
}
```

Rep: safe or exposed?

```
export class HallOfFame {
  private readonly records: Map<Sudoku, number> = new Map();

  /** @param time >= 0 */
  public addRecord(puzzle: Sudoku, time: number): void {
    const record = this.records.get(puzzle);
    if (record === undefined || time < record) {
      this.records.set(puzzle, time);
  }

  /** @param puzzle must have been added */
  public getRecord(puzzle: Sudoku): number {
    return this.records.get(puzzle) ?? assert.fail();
  ...
}
```