Example solutions (spring 2020)

This is a subset of the questions that were asked in Java on 6.031 Spring 2020 Quiz 2. They have been translated to TypeScript here.

The code for this quiz has two ADTs.

`Card` represents an immutable standard playing card. A card has a *rank* (2-10, jack, queen, king, ace) and a *suit* (clubs, diamonds, hearts, or spades).

`Deck` represents a mutable ordered set of at most 52 different playing cards.

The code is provided at the bottom of this page.

---

1. (9 points) Write a rep for `Deck`, and write its constructor. Don't implement any other operations for `Deck`. You may use the `Card.ALL_52_CARDS` constant.

```
export class Deck {

    private readonly cards: Array<Card> = [...Card.ALL_52_CARDS];

    public constructor() {
    }

}
```

---

2. (8 points) Write the abstraction function for your `Deck` rep:

> AF(cards) = the deck of playing cards consisting of the cards in `cards` in the same order, where cards[0] is the top of the deck and cards[cards.length-1] is the bottom of the deck.

---

3. (8 points) Write the rep invariant for your `Deck` rep:

> cards.length <= 52
> all elements of cards are pairwise distinct

---

4. (8 points) Observe that the return type of `Deck.draw()` is `Optional<Card>`, an abstract datatype that either has a reference to a Card, or has no value.

Suppose you are going to implement the generic `Optional<E>`. Write a datatype definition for `Optional<E>`. Your datatype definition should have two variants.

> `Optional<E> = Empty + Present(e:E)`

---

5. (9 points) Write the interface and class definitions for your datatype definition. Include the reps, but **no methods or constructors**. Your answer should consist of Typescript code from which all methods and constructors (and their signatures and bodies and associated comments) have been deleted. Don't write more than the 10 lines this box allows.

```
interface Optional<E> { }

class Empty<E> implements Optional<E> { }

class Present<E> implements Optional<E> {
    private readonly e: E;
}
```

6. (9 points) We want to add an operation `orElse` to `Optional<E>`, such that (for example) `deck.draw().orElse(aceOfSpades)` returns the top card on the deck if the deck is nonempty and `aceOfSpades` if the deck is empty.

Define `orElse` as a function using mathematical notation, with one case for each variant of your datatype definition. Your answer should be two lines long, and should not be written in Typescript code. (An example of a function written in mathematical notation is f(x) = x+1.)

> orElse(Empty, defaultValue) = defaultValue
> orElse(Present(e), defaultValue) = e

---

7. (8 points) A playing card can be described by a 2- or 3-character string such as "10C" for the ten of clubs, or "JH" for the jack of hearts. Write a grammar for these string representations of playing cards. Your grammar should have three nonterminals: `card`, `suit`, and `rank`.

> card := rank suit;
> rank ::= [AJQK2-9] | '10';
> suit ::= [CDHS];

---

8. (9 points) Suppose `Deck` has a `filter` operation that keeps cards matching a predicate, and discards cards that don't. Write a mathematical type signature for a `filter` **producer** operation of `Deck`. (An example of a mathematical type signature is `factorial`: `int` → `int`.)

> filter: Deck x (Card -> Boolean) -> Deck

---

9. (8 points) Write a Typescript instance method signature for `filter` as a **mutator** operation of `Deck`.

> public filter(predicate: (card: Card) => boolean): void;

Deck

```
/** Represents a mutable deck of distinct playing cards. */
export class Deck {

    // rep: ...

    // creates a deck starting with 52 unique playing cards in unspecified order
    public constructor() {
        ...
    }

    // removes and returns top card from deck, unless deck is empty
    public draw(): Optional<Card> {
        ...
    }

    // moves card so that it is nth from the top of this deck
    // (i.e. move(card, 0) makes card the top card of the deck).
    // No effect if card is not found in this deck.
    public move(card: Card, position: number): void {
        ...
    }
```

```
}
```

Card

```
/** Represents an immutable threadsafe standard playing card. **/
export class Card {
    // immutable set containing all the standard playing cards
    public static readonly ALL_52_CARDS: ReadonlySet<Card> = ...;

    // operations: ...
}
```