Day 19

State of the Sleigh

The Story

"Last year, someone called `take_off` mid-flight, and we ended up performing air donuts over Siberia. Never. Again."

"It was kind of cool, though." Prancer muttered.

The Story

Santa whirled. "Cool? COOL? I had to explain to a reindeer rescue hotline why Blitzen was tangled in GPS satellites! No more cowboy coding. We're doing this with the type state pattern."

Bernard hesitated. "Uh, what's wrong with just adding a check for `is_flying`?"

The Story

Santa's glare could have melted the North Pole. "Oh, brilliant idea, Bernard. Let's just trust runtime checks! No. The sleigh will know its state: parked, flying, or landed. You call `take_off` while flying? The compiler won't even let you. No crashes, no ISS collisions, no lawsuits."

The Story

Blitzen snorted. "Sounds over-engineered."

"BLITZEN, do you want to be the first reindeer to test the emergency eject system? Get to work. I want this sleigh state-safe by tonight!"

Your Mission

We don't want to end up with a sleigh in the wrong state again and we certainly don't want the `take_off` method to be available when the sleigh is already flying.

And Santa doesn't want any runtime checks, so it must all be checked at compile time.

Your Mission

Here's what you need to do:

- Finish the `Sleigh` struct to represent the state of the sleigh.
- The `Sleigh` struct must have 3 states `Empty`, `Ready` and `Flying`.

Methods

There are a few methods we want to define for each state:

Empty State

- `new()` an associated function that creates a new `Sleigh` in the `Empty` state.
- `load()` a method that transitions the `Sleigh` from `Empty` to `Ready`.

Methods

Ready State

- `take_off()` a method that transitions the `Sleigh` from `Ready` to `Flying`.
- `unload()` a method that transitions the `Sleigh` from `Ready` to `Empty`.

Flying State

- `land()` a method that transitions the `Sleigh` from `Flying` to `Ready`.

Hints

If you're stuck or need a starting point, here are some hints to help you along the way!

- Create three structs for each state `Empty`, `Ready` and `Flying`.

```
pub struct Empty;
pub struct Ready;
pub struct Flying;
```

Hints

- You can create a generic struct that represents the state of the sleigh. e.g.

```
struct Sleigh<State> {
    // State must be used
}
```

Hints

- Use PhantomData to ensure that the state is used in the struct without consuming any memory.

```
use std::marker::PhantomData;

struct Sleigh<State> {
    _state: PhantomData<State>,
}
```

Hints

- Implement the methods for each state. e.g.

```
impl Sleigh<Empty> {
    pub fn new() → Self {
        Self { state: PhantomData }
    }

    pub fn load(self) → Sleigh<Ready> {
        Sleigh { state: PhantomData }
    }
}
```