Day 16

Santa's Perfect Struct

The Story

Santa stomped into Bernard's workshop, snatching a mug of cocoa off the table without asking. He squinted at the code from yesterday glowing on the monitor. "Ah, there it is! The Wrapped status. Beautiful work, Bernard. Gifts wrapped and ready to go."

Bernard gave a tired nod, his tools still in hand. "Yeah, it’s done. So what now?"

The Story

Santa's eyes gleamed as he set the mug down. "Now... we distribute those gifts. And I know exactly how. Bernard, I need a struct. A unit struct. Call it `Santa`."

Bernard sighed. "Go on..."

The Story

"One method," Santa said, his hands animated like he was pitching a startup. "Two generics: one for the recipients—kids, elves, reindeer. The other for the gifts—`KidsGift`, `ElvesGift`, `ReindeerGift`."

"Two generics?" Bernard raised an eyebrow. "Why not just one?"

The Story

Santa grinned, his beard twitching with excitement. "Trait bounds, Bernard. Nothing gets my sleigh flying like a beautifully constrained generic. where clauses are Christmas magic."

Bernard groaned. "Fine. Two generics. Single method. Anything else?"

The Story

Santa waved dismissively. "You know the drill—keep it under wraps. No one can know until it’s perfect." He leaned in closer, eyes twinkling. "And by perfect, I mean it better compile the first time."

As Santa vanished back into the snowy chaos, Bernard muttered, "He loves trait bounds more than Christmas itself."

Your Mission

It's clear that Santa is now obsessed with trait bounds, so we're gonna see a lot of it here.

Here's what you gotta do:

- Since we use trait bounds, we need to define a new trait named `Giftable` for entities that can receive gifts. `Kid`, `Elf`, and `Reindeer` should implement this trait.

Your Mission

- `Giftable` should have a method named `receive_gift` it should set the `gifted` field to `true`.
- Update the `Gift` trait implementation for `KidsGift`, `ElvesGift`, and `ReindeerGift` to include a method `is_wrapped()` which returns a `bool` to check if the gift is wrapped.

Your Mission

- Update the `Santa` struct to have a method named `give_gift` that accepts two arguments: a recipient which could be any of `Kid`, `Elf`, or `Reindeer`, and a gift which could be any of `KidsGift`, `ElvesGift`, or `ReindeerGift`.

Your Mission

- Make sure you return a `Result<(), Box<dyn Error>>` , if the gift is not wrapped, return an error message.
- You can also have a look at the `main` function at the end of the code to see how it works in action.

Hints

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

- The `receive_gift` method should take a mutable reference to the `self` object since it mutates a value.

```
pub trait Giftable {
    fn receive_gift(&mut self);
}
```

Hints

- To define a function with more than one trait bound you can use the following syntax:

```rust
impl Santa {
    pub fn give_gift<R, G>(&self, recipient: &mut R, gift: &G) -> Result<(), Box<dyn Error>>
    where
        R: Giftable,
        G: Gift,
    {
        // your code here
    }
}
```