Day 15

Santa's Generic Obsession

The Story

"Bernard! Did I tell you? Trait bounds are the greatest thing since reindeer! The Display trick you pulled yesterday? Pure genius. I'm hooked."

Bernard shuffled in, already regretting showing Santa Display. "What now?"

The Story

"The gifts won't wrap themselves! I need a function—generic, reusable, beautiful. We slap on some bounds, and boom, wrapping perfection!"

Bernard peeked at the screen. "Santa, you've got mutable and immutable borrows fighting each other. And what's with this raw pointer?"

The Story

Santa grinned. "Bounds solve everything, Bernard! I'll slap a Display here, a Debug there—easy!"

The IDE chimed angrily. "Cannot borrow self as mutable while also borrowed as immutable."

The Story

Bernard sighed. "Santa, that's the compiler telling you no."

Santa glared at the screen. "Warm up the sled. I'm giving the borrow checker a piece of my mind."

Your Mission

What you need to do is simple, Santa has written a function `prepare_gift`, but it doesn't work quite yet. The function must accept anything that implements the `Display` and `Gift` trait. And the `Gift` trait must have a method `wrap` that mutates the `is_wrapped` field to `true`.

Your Mission

Here's what you gotta do:

- Add the new field to the structs `is_wrapped: bool`.
- Finish the `Gift` trait definition, add a method called `wrap`.

Your Mission

- Implement the `Gift` trait for `KidsGift`, `ElvesGift`, and `ReindeerGift`. Using the `wrap` method, it should set `is_wrapped` to `true`.
- Update the `prepare_gift` function signature to accept any type that implements the `Display` and `Gift` traits.

Hints

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

- The `Gift` trait should have a method `wrap` that takes a mutable reference of `self`. e.g.

```
pub trait Gift {
    fn wrap(&mut self);
}
```

Hints

- To make a function accept any type that implements multiple traits, you can use the `+` operator. e.g.

```
fn prepare_gift<T: fmt::Display + Gift>(gift: &mut T) {
    // code remains the same
}
```

- Make sure the argument takes a mutable reference `&mut T` to be able to call the wrap method since `wrap` also mutates the struct.