rust traits

sipe

traits란?

- **특성**이란 뜻
- **동작**을 정의하는 개념
  - java의 **interface**와 유사
- 공통 기능을 추상화할 수 있음
  - 오버라이딩을 통해 구현 가능하며, 기본 구현을 사용하는 것 또한 가능

trait 정의

```rust
pub trait Summary {
    fn summarize(&self) -> String;
}
```

- 해당 trait를 보유한 모든 타입에, 정확히 같은 메서드 시그니처가 구현되기를 강제

trait 구현

```rust
pub struct NewsArticle {
    pub headline: String,
    pub location: String,
    pub author: String,
    pub content: String,
}

impl Summary for NewsArticle {
    fn summarize(&self) -> String {
        format!("{}, by {} ({})", self.headline, self.author, self.location)
    }
}

pub struct Tweet {
    pub username: String,
    pub content: String,
    pub reply: bool,
    pub retweet: bool,
}

impl Summary for Tweet {
    fn summarize(&self) -> String {
        format!("{}: {}", self.username, self.content)
    }
}
```

- impl {trait} for {struct} 키워드를 통해 trait 구현 가능

trait 사용

```
use aggregator::{Summary, Tweet};

fn main() {
    let tweet = Tweet {
        username: String::from("horse_ebooks"),
        content: String::from(
            "of course, as you probably already know, people",
        ),
        reply: false,
        retweet: false,
    };

    println!("1 new tweet: {}", tweet.summarize());
}
```

- 구현 타입 뿐만 아니라, trait도 스코프에 가져와야 사용할 수 있음에 주의

trait 제약

- 외부 타입에 외부 trait를 구현하는 것을 불가능
  - 예를 들어, 각자 표준 라이브러리인 Vec<T>에 Display trait 재정의는 불가
- 일관성을 유지하기 위해서
  - 두 크레이트가 동일한 타입에 동일한 트레이트를 재구현하게 되는 것을 방지
  - 컴파일러가 어떤걸 사용해야 할지 모르기 때문

trait 기본 구현

```
pub trait Summary {
    fn summarize(&self) -> String {
        String::from("(Read more...)")
    }
}
```

- trait에 메서드 내용 정의 시, 기본 구현 적용
  - `impl Summary for NewsArticle {}`과 같이, 해당 메서드를 재정의하지 않으면 기본 구현 사용
  - 기본 구현을 재정의 시, 기본 구현은 사용 불가

trait bound

```
pub fn notify<T: Summary>(item: &T) {
    println!("Breaking news! {}", item.summarize());
}
```

- 특정 trait의 generics 개념
  - 해당 trait을 구현한 타입만 매개변수로 허용됨

trait bound

```
pub fn notify(item: &impl Summary) {
    println!("Breaking news! {}", item.summarize());
}
```

- 시그니처를 impl {trait}으로 축약 가능

```
pub fn notify(item1: &impl Summary, item2: &impl Summary) {
```

```
pub fn notify<T: Summary>(item1: &T, item2: &T) {
```

- 상황에 따라 더 깔끔한 문법이 존재하므로, 잘 골라 사용

trait bound 관련 유용한 문법

```rust
pub fn notify(item: &(impl Summary + Display)) {
```

```rust
pub fn notify<T: Summary + Display>(item: &T) {
```

- + 연산자를 통해 여러 trait를 허용하는 bound 정의 가능

```rust
fn some_function<T, U>(t: &T, u: &U) -> i32
where
    T: Display + Clone,
    U: Clone + Debug,
{
```

- where 키워드를 통해 trait bound 가독성 강화 가능

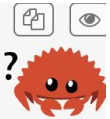
trait 구현 타입 반환

```
fn returns_summarizable() -> impl Summary {
    Tweet {
        username: String::from("horse_ebooks"),
        content: String::from(
            "of course, as you probably already know, people",
        ),
        reply: false,
        retweet: false,
    }
}
```

- impl {trait}과 같이 반환 타입 부착
- 해당 trait을 구현한 모든 타입 반환 가능

trait 구현 타입 반환

```rust
fn returns_summarizable(switch: bool) -> impl Summary {
    if switch {
        NewsArticle {
            headline: String::from(
                "Penguins win the Stanley Cup Championship!",
            ),
            location: String::from("Pittsburgh, PA, USA"),
            author: String::from("Iceburgh"),
            content: String::from(
                "The Pittsburgh Penguins once again are the best \
                 hockey team in the NHL.",
            ),
        }
    } else {
        Tweet {
            username: String::from("horse_ebooks"),
            content: String::from(
                "of course, as you probably already know, people",
            ),
            reply: false,
            retweet: false,
        }
    }
}
```



- but, 한 메서드에서는 하나의 타입만 반환해야 함
  - 컴파일러 내 구현 방식으로 인해 허용되지 않으며, 추후 이 방식을 구현할 수 있는 방법 배울 예정

trait 포괄 구현

```
impl<T: Display> ToString for T {
    // --생략--
}
```

- Display trait을 구현하는 모든 타입에게 ToString trait을 부여하는 예제
  - 이를 포괄 구현(*blanket implementations*)이라 칭함

```
let s = 3.to_string();
```

- Display trait을 구현하는 모든 타입이 to_string()을 호출할 수 있는 이유