
RUST

Generički tipovi, Trait

Generički tipovi

- Kreiranje definicija potpisa funkcija, struktura ili enuma koje onda možemo da koristimo sa mnogo različitih konkretnih tipova podataka.

Funkcija

- Generički tip stavljamo u potpis funkcije gde obično specificiramo tipove podataka parametara i povratne vrednosti.
- Fleksibilniji kod i pruža više funkcionalnosti

Funkcija

```
fn largest_i32(list: &[i32]) -> &i32 {  
}
```

```
fn largest_char(list: &[char]) -> &char {  
}
```



```
fn largest<T>(list: &[T]) -> &T {  
}
```

Struktura

- Strukture mogu da koriste parametre generičkog tipa u jednom ili više polja koristeći sintaksu <>.

Struktura

```
struct Point<T> {  
    x: T,  
    y: T,  
}
```

```
fn main() {  
    let integer = Point { x: 5, y: 10 };  
    let float = Point { x: 1.0, y: 4.0 };  
}
```

```
struct Point<T> {  
    x: T,  
    y: T,  
}
```

```
fn main() {  
    let wont_work = Point { x: 5, y: 4.0 };  
}
```

Enum

```
enum Option<T> {  
    Some(T),  
    None,  
}
```

```
enum Result<T, E> {  
    Ok(T),  
    Err(E),  
}
```

Metode

```
struct Point<T> {  
    x: T,  
    y: T,  
}
```

```
impl<T> Point<T> {  
    fn x(&self) -> &T {  
        &self.x  
    }  
}
```


Generički tipovi

- Korišćenje generičkih tipova neće učiniti da vaš program radi ništa sporije nego što bi to bilo sa konkretnim tipovima.
- Monomorfizacija koda upotrebom generičnosti u vreme kompajliranja.
 - Proces pretvaranja generičkog koda u specifičan kod popunjavanjem konkretnih tipova koji se koriste prilikom kompajliranja.
 - Kompajler pregleda sva mesta na kojima se poziva generički kod i generiše kod za konkretne tipove sa kojima se generički kod poziva.

Generički tipovi

```
let integer = Some(5);
```

```
let float = Some(5.0);
```

monomorfizacija

```
enum Option_i32 {  
    Some(i32),  
    None,  
}
```

```
enum Option_f64 {  
    Some(f64),  
    None,  
}
```

```
fn main() {  
    let integer = Option_i32::Some(5);  
    let float = Option_f64::Some(5.0);  
}
```

Trait

- Definiše funkcionalnost koju određeni tip ima ili može da deli sa drugim tipovima.
- Možemo da ih koristimo da definišemo zajedničko ponašanje na apstraktan način.
- *Trait bounds* - generički tip može biti bilo koji tip koji ima određeno ponašanje.

Trait

- Ponašanje nekog tipa zavisi od metoda koje možemo pozvati nad tim tipom.
- Različiti tipovi dele ponašanje ako možemo da pozovemo iste metode za sve te tipove.
- Definisanje *trait-a* je način da se zajedno grupišu potpisi metoda kako bi se definisao skup ponašanje neophodnih za postizanje neke svrhe.

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

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

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

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

Trait

- Spoljašnje osobine ne možete da primenite na spoljašnje tipove.
 - *Pr. ne možete primeniti Display osobinu na `Vec<T>` u okviru vašeg sanduka, zato što su i osobina i struktura definisane u standardnoj biblioteci i nisu lokalni za vaš sanduk.*
- Ovo ograničenje je deo svojstva, koje se zove *koherentnost*
 - Roditeljski tip nije prisutan
 - Osigurava da kod drugih ljudi ne može da razbije vaš kod i obrnuto
 - *Pr. dva sanduka bi mogla da implementiraju istu osobinu za isti tip, a Rust ne bi znao koju implementaciju da koristi.*


Podrazumevana implementacija

- Ponekad je korisno imati podrazumevano ponašanje za neke ili sve metode u *trait-u* umesto da zahtevate implementacije za sve metode za svaki tip.

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

Trait kao parametar

Bilo koji tip koji
implementira navedenu
osobinu.

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


Trait bound

- *impl Trait* sintaksa radi za jednostavne slučajeve i predstavlja sintaktički šećer za duži oblik, *trait bound*

Trait bound

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



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

Impl Trait vs Trait bound

- *Impl Trait* sintaksa je zgodna i čini koncizniji kod u jednostavnim slučajevima, dok *trait bound* sintaksa može izraziti više složenosti

Item1 i item2 mogu da budu različitog tipa

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

Item1 i item2 moraju da budu istog tipa

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

Trait bound

- Možete specificirati i više trait bound-ova istovremeno upotrebom operatora +

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

*Item mora da implementira osobine Summary
i Display*

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

Trait bounds

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



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

Trait kao povratna vrednost

```
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,  
    }  
}
```

```
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,  
        }  
    }  
}
```

Zadatak

Kreirati strukture Film i Serija i za njih napraviti osobine koje mogu da se primene i na jednu i na drugu strukturu.