Osobine, životni vekovi, rukovanje greškama

Paralelne i distribuirane arhitekture i jezici
Računarstvo visokih performansi
Zimski semestar, školska 2024/25.
Branislav Ristić

Osobine (*Traits*)

- Trait “objekti” omogućavaju polimorfizam bez nasleđivanja.
- Trait definiše zajedničko ponašanje kroz metode.
- Trait objekti kombinuju podatke i ponašanje.
- Primer:
  - *01_traits.rs*

Osobine - Podrazumevane implementacije

- Trait-ovi mogu imati podrazumevane metode za ponašanje
- Tipovi mogu koristiti ili prilagoditi podrazumevane metode
  - Podrazumevana metoda nije dostupna prilagođenoj implementaciji
- Podrazumevane metode mogu koristiti druge trait metode
- Smanjuje potrebu za pisanjem redundantnog koda
- Primer:
  - *02_default.rs*

Osobine - Parametri

- `impl Trait` omogućava prihvatanje bilo kog tipa koji zadovoljava trait
- `T: Trait` sintaksa takođe specificira ograničenja osobina, ali na nivou tipa
  - Korišćenjem `T` forsira se posedovanja istog tipa
- Tipovi parametara mogu da implementiraju više traitova npr. `Summary + Display`
- `where` klauzula povećava čitljivost
- Primer:
  - *03_parameters.rs*

Osobine - Povratne vrednosti

- `impl Trait` skriva konkretan tip povratne vrednosti
  - npr. `-> impl Summary` za fleksibilnost tipa
- **Ograničeno na jedan konkretan povratni tip**
- Korišćenje `dyn` i pointera za dinamički pristup
  - Omogućava rad sa različitim tipovima, koji implementiraju istu osobinu, u vektoru
- Primer:
  - *04_return_val.rs*

Osobine - Uslovna implementacija metoda

- Moguće je implementirati određene metode ukoliko tip poseduje određeni trait
- *Blanket implementations*
- Provere pri kompajliranju koda
- Primer:
  - *05_cond_impl.rs*
  - *06_blanket.rs*

Životni vekovi (*Lifetimes*)

- Životni vekovi u Rustu osiguravaju da reference ostanu važeće tokom korišćenja
- Cilj životnih vekova je da spreče nevažeće reference
- U većini slučajeva, vekovi se automatski zaključuju
- Životni vekovi moraju biti eksplicitni kada reference imaju složene odnose
- Rust-ov borrow-checker detektuje probleme sa referencama tokom kompajliranja
- Primer:
  - *07_a_lifetime.rs*
  - *07_b_lifetime.rs*

Životni vekovi

- Lifetime anotacije opisuju odnose vekova referenci **bez menjanja trajanja**
- Anotacije omogućavaju funkcijama da prihvate reference sa bilo kojim vekom
- Imena vekova počinju apostrofom `'`
- Najčešće korišćeno ime za vek je `'a`
- Sintaksa:

```
&i32        // a reference
&'a i32     // a reference with an explicit lifetime
&'a mut i32 // a mutable reference with an explicit lifetime
```

Životni vekovi

- Koriste se u potpisima funkcija
  - Kao jedan od generičkih parametara
- Vraćena reference iz funkcije važi koliko i parametri
  - To jest, koliko i najkraće živeći parametar
- Primer:
  - *08_a_function.rs*
  - *08_b_function.rs*

Životni vekovi - Strukture

- Strukture sa referencama zahtevaju anotaciju životnog veka
- Životni vek reference definiše vek strukture
  - Struktura ne može trajati duže od referencirane vrednosti
- Anotacija osigurava memorijsku bezbednost
  - Životni vek štiti od grešaka jer referencira podatke koji traju dovoljno dugo.
- Primer:
  - *09_struct.rs*

Automatski životni vekovi

- Skup pravila kompajlera za automatsko dodeljivanje životnih vekova
  - Smanjuje potrebu za eksplicitnim anotacijama.
- Pravila:
  - **Pravilo 1**: Svaki ulazni parametar dobija svoj životni vek.
  - **Pravilo 2**: Ako postoji samo jedan ulazni životni vek, on se dodeljuje svim izlaznim referencama.
  - **Pravilo 3**: Za metode, ako ima više životnih vekova, životni vek self se dodeljuje svim izlaznim referencama.
- Ako pravila ne mogu da odrede sve životne vekove
  - Potrebne su ručne anotacije
- Primer:
  - *10_elision.rs*

Životni vekovi - Metode

- Implementacija metoda sa životnim vekom koristi sličnu sintaksu kao generički tipovi.
- Životni vek za polja strukture navodi se posle `impl`, jer je deo njenog tipa.
- U `impl` bloku, životni vekovi metoda mogu biti vezani za polja strukture ili nezavisni.
- Rust često automatski dodaje životni vek, pa eksplicitna anotacija nije uvek potrebna.
- Primer:
  - *11_method.rs*

Životni vekovi - *static*

- `'static` životni vek
  - Znači da referenca traje tokom celog programa.
- Svi string literali imaju `'static` životni vek,
  - npr. `let s: &'static str = "I have a static lifetime.";`
- Tekst string literala se čuva direktno u binarnom kodu.
- Uobičajene greške:
  - Greške koje predlažu 'static često nastaju zbog neusaglašenih životnih vekova ili "visećih" referenci.
- Primer:
  - *12_static.rs*

Životni vekovi - Generici, osobine

- Moguće je za specificirati životne vekove i za strukture koje prihvataju:
  - Proizvoljan tip
  - Osobine
- Primer:
  - *13_generics.rs*
  - *14_foo.rs*

Rukovanje greškama

- Rust ne poseduje izuzetke.
- Umesto toga poseduje greške.
- Greške se dele na:
  - Nadoknadive
    - Otvaranje datoteke koja ne postoji.
  - Nenadoknadive
    - Pristup vektoru na indeksu van granica.

Nenadoknadive greške

- Makro panic! koristi se za fatalne greške.
- Panikom se zaustavlja program i čisti stek.
- Podrazumevano ponašanje je "unwinding" i zatvaranje programa.
- Opcija "abort" završava program bez čišćenja memorije.
- Backtrace omogućava praćenje tačke izazivanja panike.
- Primer:
  - *15_panic.rs*

Nadoknadive greške

- Većina grešaka nije razlog za potpuni završetak programa
- Umesto se koristi enum `Result<T, E>`
- Na primer, neuspešno otvaranje datoteke
- Primer:
  - *16_error.rs*

Error matching

- Rust omogućava načine da se odredi do koje greške je došlo
- Pošto je `Error` struktura u okviru Rust-a, moguće je vršiti poređenje po vrsti
- Primer:
  - *17_error_matching.rs*

unwrap

- Metoda `unwrap` prestavlja skraćenicu implementrirana nad `Result<T, E>` kao:
  - `match`
    - U `Ok` slučaju
      - Vraća otpakovanu vrednost
    - U `Error` slučaju
      - Izaziva panic
- Primer:
  - *18_a_unwrap.rs*
  - *18_b_unwrap.rs*

expect

- Metoda koja suštinski radi kao `unwrap`
  - U slučaju `panic`-a koristi
    - Vrednost prosleđenog parametra
- Primer:
  - *19_expect.rs*

Propagacija grešaka

- Ukoliko se greška dogodi prilikom poziva funkcije unutar druge funkcije
- Moguće je “ne obraditi” grešku, već proslediti je naviše
  - Mora biti u okviru funkcije koja vraća Result
- Operator `?`
- Primer:
  - *20_a_propagate.rs*
  - *20_b_propagate.rs*
  - *20_c_propagate.rs*
  - *20_d_propagate.rs*

Option i ?

- Korišćenje operatora `?` sa `Option` vrednostima je takođe moguća
  - U slučaju `None`
    - Biće vraćena None iz cele funkcije
  - U slučaju `Some`
    - Biće vraćena vrednost u okviru Some
- Nije moguće koristiti:
  - `option?` u okviru funkcije koja vraća `Result`
  - `result?` u okviru funkcije koja vraća `Option`
- U tom slučaju koristiti metode `ok` ili `ok_or`, ili neke od srodnika
- Primer:
  - *21_option.rs*
  - *22_option_result.rs*

Zadatak 1.

- Kreirati strukturu Pair koja sadrži dve reference, &str i &i32, svaka sa sopstvenim životnim vekom.
- Zatim implementirati dve metode:
  - get_first - vraća &str
  - get_second - vraća &i32
- Testirati program.

Zadatak 2.

- Kreirati osobinu Printable, koja sadrži metodu print.
- Zatim definisati dve strukture:
  - Krug sa poljem radijusa.
  - Pravougaonik sa poljima širine i visine.
- Implementirati Printable za svaku strukturu. Svaki metod štampanja treba da prikaže detalje strukture na jedinstven način.
- Testirati program.

Izvori

- Rust Community. “The Rust Programming Language - the Rust Programming Language.” Rust-Lang.org, 2018, doc.rust-lang.org/book/.
- Crichton, Will. “Experiment Introduction - the Rust Programming Language.” Brown.edu, rust-book.cs.brown.edu/.
- Rust Community. “Tour of Rust - Let’s Go on an Adventure!” Tourofrust.com, tourofrust.com/.
- Rust Team. “Rust Programming Language.” Rust-Lang.org, 2018, www.rust-lang.org/.

Osobine, životni vekovi, rukovanje greškama

Paralelne i distribuirane arhitekture i jezici
Računarstvo visokih performansi
Zimski semestar, školska 2024/25.
Branislav Ristić