Rust

Upravljanje velikim projektima

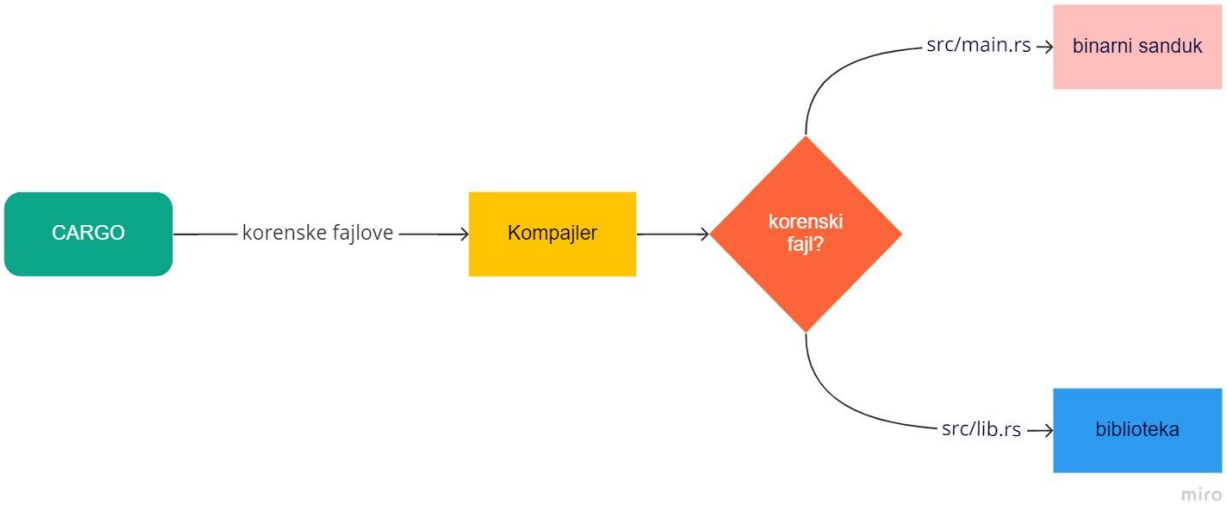
Sanduk

- Najmanja količina koda koju Rust kompajler istovremeno može da razmatra
  - Ako pokrenete *rustc* komandu i njoj prosledite *.rs* fajl, kompajler će taj fajl da posmatra kao sanduk.
- Može da sadrži više modula, a moduli mogu biti definisani u različitim fajlovima koji se kompajliraju sa sandukom.
- Vrste:
  - *Binarni* - Poseduje *main* funkciju i kompajliranjem se dobija izvršni fajl, koji možete da pokrenete.
  - *Biblioteka* - Nema *main* funkciju i ne kompajlira se u izvršni fajl. Definiše funkcionalnosti namenjene za deljenje sa drugim projektima.
    - *rand* biblioteka

Paket

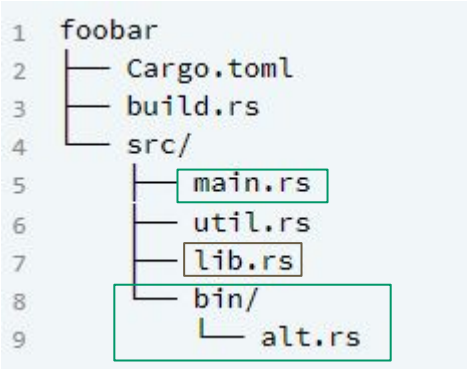
- Skup jednog ili više sanduka koji omogućavaju skup funkcionalnosti.
- Sadrži datoteku *Cargo.toml* koja definiše kako se paket kreira.
- Paket može da sadrži koliko god želite binarnih sanduka, ali isključivo samo jednu biblioteku. Mora da sadrži najmanje jedan sanduk, bilo da je to biblioteka ili binarni sanduk.
    - Paket može ima više binarnih sanduka tako što datoteke postavite u folder *src/bin,* svaki fajl u ovom folderu će biti zaseban binarni sanduk.
- Za kreiranje paketa se koristi komanda:
    - ***cargo new***

Sanduk i paket



Sanduk i paket

Koliko sanduka sadrži paket foobar?



Moduli

- Omogućavaju:
  - organizaciju koda unutar sanduka radi čitljivosti i lake ponovne upotrebe,
  - kontrolisanje privatnost stavki
    - Kod u okviru modula je privatan

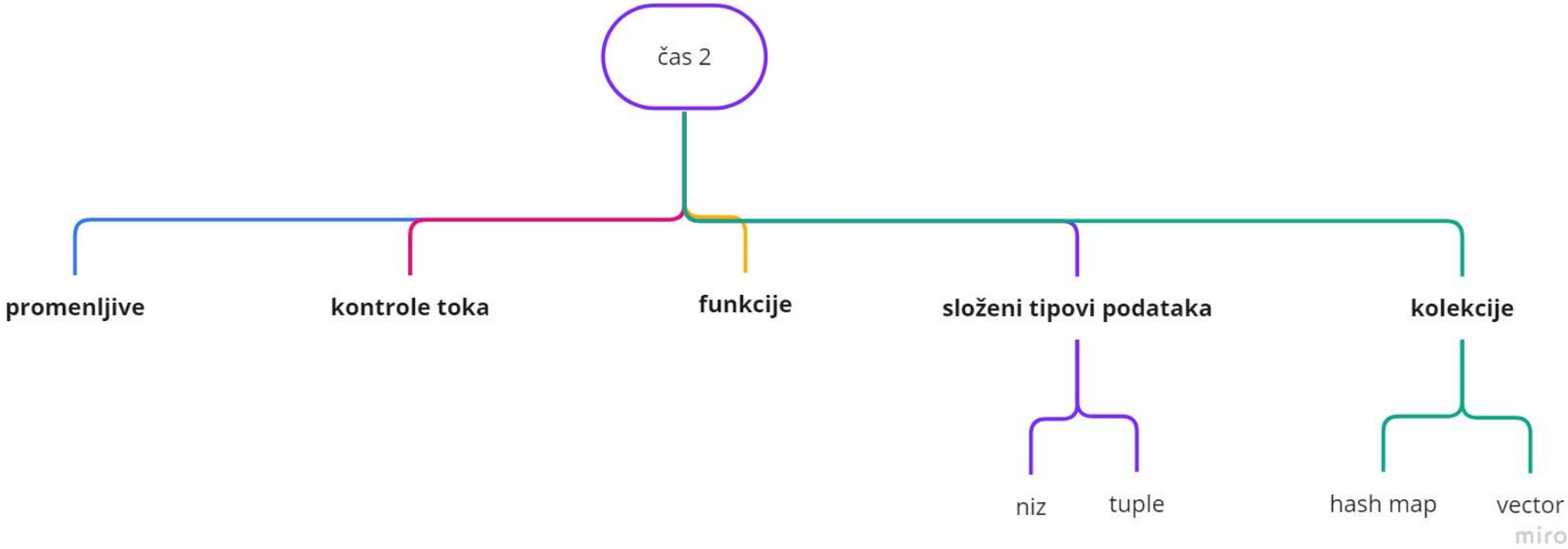
Moduli

1. Koren sanduka
   a. Prvo se gleda da li je sanduk binarni ili je biblioteka
2. Deklarisanje modula
   a. U okviru korena sanduka možete da definišete nove module, upotrebom ključne reči *mod.* Kompajler će pokušati modul da pronađe:
      i. *inline* - na mestu definisanje modula, odmah posle izraza *mod module_name*
      ii. U fajlu koji se zove isto kao modul i nalazi se u *src* folderu (*src/module_name*)
      iii. U fajlu *src/module_name/mod.rs*
3. Deklarisanje podmodula
   a. U bilo kojoj datoteci osim u korenu sanduka možete da definišete podmodule. Kompajler će podmodul pokušati da pronađe:
      i. *inline* - na mestu definisanje modula, odmah posle izraza *mod module_name*
      ii. U fajlu *src/module_name/submodule_name*
      iii. U fajlu *src/module_name/submodule_name/mod.rs*

Moduli

4. Putanja do koda u modulima
    a. Jednom kada je modul deo projekta, njegov kod možete koristiti bilo gde u projektu sve dok pravila privatnosti to dozvoljavaju, koristeći putanju do koda.
        i. Primer: *crate::module::submodule::item*
5. Privatno vs javno
    a. Kod modula je podrazumevano privatan od njegovih roditeljskih modula.
    b. Da biste modul učinili javnim treba da upotrebite *pub mod* prilikom definisanja modula.
    c. Stavke u okviru modula možete da učinite javnim upotrebom *pub* pre njihove definicije.
6. Ključna reč *use*
    a. U okviru opsega, kreira prečice do stavki kako bi se smanjilo ponavljanje dugih putanja.

Moduli - primer



Putanje do stavki modula

- Apsolutna putanja - puna putanja koja počinje od korenskog direktorijuma sanduka
  - Ako je kod iz ekstrenog sanduka onda počinje sa imenom sanduka,
  - A ako je kod iz trenutnog sanduka onda počinje sa *crate*
- Relativna putanja - počinje od trenutnog modula i koristi *self, super* i identifikatore trenutnog modula
- Obe putanje kao separator između identifikatora koriste “::”.

Putanje do stavki modula

```
mod front_of_house {
    mod hosting {
        fn add_to_waitlist() {}
    }
}

pub fn eat_at_restaurant() {
    // Absolute path
    crate::front_of_house::hosting::add_to_waitlist();

    // Relative path
    front_of_house::hosting::add_to_waitlist();
}
```

Izlaganje putanje pomoću ključne reči *pub*

- Kod sa prethodnog slajda se neće kompajlirati, zato što je modul podrazumevano privatan i njemu ne možemo da pristupimo → moramo upotrebi ključnu reč *pub* kako bi modul učinili javnim
- Ako stavimo da je modul *pub* onda možemo da mu pristupimo, ali njegov sadržaj je i dalje privatan → pravljenje modula javnim ne čini njegov sadržaj javnim
- *Pub* samo omogućava da se kod u njegovim prethodnim modulima odnosi na njega a ne i da pristupi njegovim unutrašnjim delovima.

Ključna reč *pub*

```
mod front_of_house {
    pub mod hosting {
        pub fn add_to_waitlist() {}
    }
}

pub fn eat_at_restaurant() {
    // Absolute path
    crate::front_of_house::hosting::add_to_waitlist();

    // Relative path
    front_of_house::hosting::add_to_waitlist();
}
```

Zadatak

Da li se sledeći kod kompajlira?

```
pub mod foo {
    fn a() { println!("a"); }
    mod bar {
      pub fn b() { println!("b"); }
    }
}
fn main() {
    foo::bar::b();
}
```

Odgovor: Ne, problem je na liniji 9.

Zadatak

Da li se sledeći kod kompajlira?

```
pub mod foo {
    pub mod bar {
      pub fn b() { println!("b");  }
    }
    pub fn a() { bar::b(); }
  }
  fn main() {
    foo::a();
  }
```

Odgovor: da, ispisaće “b”.

Relativna putanja i *super*

- Omogućava nam da definišemo relativnu putanju koja počinje u roditeljskom modulu, a ne u trenutnom modulu ili korenu sanduka.
- Omogućava nam da referenciramo stavku za koju znamo da se nalazi u roditeljskom modulu, što može da olakša preuređivanje stabla modula kada je modul blisko povezan sa roditeljem, ali bi roditelj jednog dana mogao biti premešten na drugo mesto u stablu.

Relativna putanja i *super*

```
fn deliver_order() {}

mod back_of_house {
    fn fix_incorrect_order() {
        cook_order();
        super::deliver_order();
    }

    fn cook_order() {}
}
```

Javna struktura i enum

- *Pub* sa strukturama
  - Ako ga koristimo pre definicije strukture onda će ona biti javna, ali njena polja su i dalje privatna
  - Svako polje u strukturi može da bude javno ili ne od slučaja do slučaja
- *Pub* sa enumom
  - Sve vrednosti enuma su javne ako enum označimo sa *pub*

```rust
mod back_of_house {
    pub struct Breakfast {
        pub toast: String,
        seasonal_fruit: String,
    }
    impl Breakfast {
        pub fn summer(toast: &str) -> Breakfast {
            Breakfast {
                toast: String::from(toast),
                seasonal_fruit: String::from("peaches"),
            }
        }
    }
}

pub fn eat_at_restaurant() {
    // Order a breakfast in the summer with Rye toast
    let mut meal = back_of_house::Breakfast::summer("Rye");
    // Change our mind about what bread we'd like
    meal.toast = String::from("Wheat");
    println!("I'd like {} toast please", meal.toast);

    meal.seasonal_fruit = String::from("blueberries");
}
```

Zadatak

Da li se sledeći kod kompajlira?

```
pub mod a {
    pub mod b {
      pub fn f() { println!("b1"); }
      pub mod c {
        pub fn f() { println!("c1"); }
      }
    }
    pub fn entry() { super::b::c::f(); }
  }
  pub mod b {
    pub fn f() { println!("b2"); }
    pub mod c {
      pub fn f() { println!("c2"); }
    }
  }
  fn main() {
    crate::a::entry();
  }
```

Odgovor: Da, ispisaće “c2”.

Zadatak

Da li se sledeći kod kompajlira?

```
pub mod point {
    #[derive(Debug)]
    pub struct Point(i32, i32);
    impl Point {
      pub fn origin() -> Self { Point(0, 0) }
    }
  }
  fn main() {
    let mut p = point::Point::origin();
    p.0 += 1;
    println!("{p:?}");
  }
```

Odgovor: Ne.

Ključna reč *use*

- Koristi se da kreira prečicu do putanje, a zatim da koristimo kraće ime u opsegu
- Kada unosite strukture, enume i druge stavke sa ključnom rečju *use*, pravilno je navesti punu putanju.
- Izuzetak:
  - Unos dve stavke sa istim imenom u opseg, nije dozvoljen

Ključna reč *use*

```
mod front_of_house {
        pub mod hosting {
                pub fn add_to_waitlist() {}
        }
}

pub fn eat_at_restaurant() {
        // Absolute path
        crate::front_of_house::hosting::add_to_waitlist();
        // Relative path
        front_of_house::hosting::add_to_waitlist();
}
```

```
mod front_of_house {
    pub mod hosting {
        pub fn add_to_waitlist() {}
    }
}

use crate::front_of_house::hosting::add_to_waitlist;

pub fn eat_at_restaurant() {
    add_to_waitlist();
}
```

Upotreba ključne reči *as*

- Koristi se za uvođenje stavki sa istim imenom u isti opseg upotrebom ključne reči *use*
- Navodi se posle putanje, a zatim se posle *as* navede lokalno ime (alias) koje se kasnije koristi

```
use std::fmt;
use std::io;

fn function1() -> fmt::Result {
    // --snip--
}

fn function2() -> io::Result<()> {
    // --snip--
}
```

```
use std::fmt::Result;

use std::io::Result as IoResult;

fn function1() -> Result {

    // --snip--

}

fn function2() -> IoResult<()> {

    // --snip--

}
```

Upotreba eksternih paketa

- Ako hoćete da koristite neke funkcionalnosti koje se nalaze u javnim bibliotekama, neophodno je da tu biblioteku uvezete u projekat na sledeći način:
  - Dodate naziv i verziju biblioteke u *dependencies* sekciju.
  - Upotrebom *use* izraza uvedete željenu stavku u opseg u kojem želite da je koristite.

Upotreba ugnjezdenih izraza za čišćenje od *use* iraza

- Ako koristite više stavki koje su definisane u istom sanduku ili istom modulu, navođenje svake stavke posebno će biti zamorno, ali zato možete upotrebiti skraćeni *use* iskaz i u okviru jednog *use* iskaza navesti više stavki koje želite da uvezete

```
// --snip--

use std::cmp::Ordering;

use std::io;

// --snip–
---------------------------
// --snip--

use std::{cmp::Ordering, io};

// --snip--
```

Glob operator

- Ako želite da uvezete sve javne stavke koje su definisane na nekoj putanji u opseg, onda možete da koristite globalni operator *

Razdvajanje modula u različite fajlove

- Svaki modul možete da napišete u posebne fajlove
- Upotrebom ključne reči *mod* treba da učitate datoteku u svoje stablo modula.