



RUST

Package - Crate -Module

Kendi Paketini Oluşturmak ve Kullanmak



Neden Kullanılır?

Büyük programlar yazarken, kodu düzenlemek daha önemli bir hale gelir

Fonksiyonları gruplandırmak, programı farklı özelliklere ayırmak, aradığınız kodu nerede bulacağınızı, nasıl değiştireceğinizi daha kolay hale gelir.

Projeyi genişletmek kolaylaşır

Projeyi yeni teknolojilere adapte etmek kolaylaşır



Moduler System

- Packages (Paketler) → Crate oluşturmanızı, test etmenizi, ve paylaşmanızı sağlayan bir özelliktir.
- Crates → Kütüphane yada çalıştırılabilir dosya üreten modül ağacıdır.
- Module ve Use → Yolların organizasyonunu, kapsamını, gizliliğini kontrol etmemizi sağlar.
- Paths → Modül, fonksiyon yada struct gibi yapıları adlandırmanın bir yoludur.



Module - mod { }

Şimdiye kadar yazdığımız programlar tek dosya olarak aslında tek bir modül yapısındadır.

Projeyi farklı dosyalara yaymak farklı modüller oluşturmaktır.



Crate

Derleyicinin bir seferde dikkate aldığı en küçük kod miktarıdır.

Cargo yerine rustc kullanarak bir dosya derlendiğinde, Rust bu dosyayı bir crate olarak değerlendirir.

Modüllerden oluşur.



Package

Paket birden fazla binary create ya da isteğe bağlı olarak bir kütüphane crate'inden oluşabilir.

Paket büyüdükçe, harici bağımlılıkları olan cratelere bölünebilir. Yani bir paket yüklendiğinde, onun çalışması için diğer harici cratelerde onunla birlikte yüklenir.



Cargo Workspace

Birlikte gelişen ve birbiri ile ilişkili bir dizi paketten oluşan çok büyük projeler için kullanılır.



Packages And Crates → Crates

Modüllerden oluşabilir ve bu modüller farklı dosyalarda da olabilir.

İki farklı yapıda crate vardır;

- binary crate → Çalıştırılabilir programlardır. Main() fonksiyonu olmak zorundadır.
- library crate → main() fonksiyonu yoktur. Çalıştırmak için derlenmezler. Farklı projelerde kullanılmak üzere hazırlanırlar.

Crate Root → Derleyici için başlangıç dosyasıdır. Crate'in kök modülünü oluşturur.



Packages And Crates → Packages

İçerisinde bir veya daha fazla crate bulunur. Bu yapı daha geniş bir kullanım sağlar.

cargo.toml dosyası bulunur. Bu cratelerin nasıl build edileceğini tanımlar.

cargo, aslında kodu build etmek için kullanılan command-line aracı için binary create'ten oluşan bir pakettir.

Paketlerde bir den fazla binary crate bulunabilir, fakat genel olarak sadece bir tane library crate bulunur.

Paketlerde en az bir crate bulunmak zorundadır ama bu binary ya da library crate olabilir.



cargo new proje-adi

Bu komut ile biz bir paket oluşturmuş oluruz. Çünkü içinde cargo.toml olduğuna dikkat ediniz.

Cargo aracı, src/main.rs 'nin crate kökü olduğunu ve bunun bir binary crate olduğunu bilir.

Eğer src/lib.rs dosyası varsa bununda library crate olduğunu bilir.

Bu kök dosyalar, library ya da binary crate oluşturmak için rustc'ye gönderilir.

New komut ile oluşturulan binary crate adı projenin adıdır. (proje-adi)

Eğer hem main.rs hem de lib.rs varsa; iki crate vardır ve isimleri paket ile aynıdır.

Pakette birden fazla binary crate bulunabilir. Bunun için dosyaları src/bin dizinine eklenmesi gerekir. Bunlar birbirinden farklı crateler olur.

cargo run --bin crate-adi → ayrı ayrı derlenirler.



Module Tanımlama → Kapsam ve Gizliliği Kontrol Eder. (paths, use, pub, as, glob operatör)

Modül Tanımlama Kuralları

Crate derleme işlemi, crate kökünden başlar. (src/lib.rs veya src/main.rs)

Modül Tanımlama → Crate root dosyasında yeni modül tanımlamak için " mod modül_adi; " yapısı kullanılır.

Derleyici modülü bulmak için sırayla şunları kontrol eder;

- Aynı dosyada " mod modül_bir { ... } " var mı?
- " src/modul-bir.rs " dosyası var mı?
- " src/modul-bir/mod.rs " dosyası var mı? (eski stil)

(Bu yapı ayrıca modülün nasıl oluşturulabileceğini de açıklamaktadır.)



Proje-adı

├── Cargo.lock

├── Cargo.toml

└── src

 ├── modül_bir

 └── alt_modul.rs

 ├── modül_bir.rs

 └── main.rs



Modül Tanımlama Kuralları

Alt modül tanımlama → Crate root dışında herhangi bir dosyada alt modül tanımlanabilir. Örneğin src/modül_bir.rs içinde " mod altmodul; " tanımlaması yapılabilir.

Bu sefer derleyici modülü aşağıdaki sıra ile arar;

- Aynı dosya da " mod altmodul { ... } "
- src/modül_bir/altmodul.rs
- src/modül_bir/altmodul/mod.rs (eski stil)



Modül Tanımlama Kuralları

Modül koduna erişmek için kullanılan yol : (Bu yapı kod içinden erişileceği zaman kullanılmaktadır. Bu şekilde uzun yazmamak için " use " kullanılır)

- modül, crate'in bir parçası olmalıdır
- gizlilik kurallarına göre erişilir.

`crate::modül-adı::altmodül-adı::Modül_İçindeki_Yapı`

`Modül_İçindeki_Yapı` → Fonksiyon, değişken, struct vb. bir şey olabilir



Modül Tanımlama Kuralları

private vs public (pub)

- Varsayılan olarak modül içindeki kodlar, modülü kapsayan üst yapılar için gizlidir, yani üst yapılardan erişilemez.
- Ulaşılması için modülün " pub mod " şeklinde oluşturulmalıdır.
- Ayrıca modül içinde erişilmesi istenen üyeler de (fonsksiyon, struct vb.) "pub" ile tanımlanmalıdırlar.



Modül Tanımlama Kuralları

use keyword → Kod içinde erişilecek olan yolu uzun yazmamak için kullanılır.

`crate::modül-adı::altmodül-adı::Modül_İçindeki_Yapı`

yerine sadece "Modül_İçindeki_Yapı" elemanı yazarak kod içinde kullanabilmek için ;

`use crate::modül_adı::altmodül_adı::Modül_İçindeki_Yapı`

tanımlaması yapılır.



Uygulama

src/main.rs

```
pub mod modul_bir;
```

src/

modül_bir.rs dosyasını oluştur

src/modül_bir.rs

```
pub mod alt_modül;
```

✓ MODULER-SYSTEM [WSL: UBUNTU-20.04]

✓ src

✓ modul_bir

alt_modul.rs 2, U

main.rs 1, U

modul_bir.rs U

> target

.gitignore U

Cargo.lock U

Cargo.toml U



Uygulama

```
src/  
    modül_bir klasörü oluştur  
    alt_modül.rs dosyası oluştur.
```

```
src/modül-bir/alt-modül.rs  
  
    struct Modul_Icindeki_Yapi {}           oluştur.
```

```
src/main.rs  
  
Use crate::modül-bir::alt-modül::Modul-Icindeki-Yapi;  
.....  
    let erisilen_eleman = Modul_Icindeki_Yapi { };
```

Uygulama sonu



Birbiri İle İlgili Kodları Modüller İle Gruplandırma

Kütüphane oluşturma → `cargo new restaurant --lib`

```
mod front_of_house {  
  mod hosting {  
    fn add_to_waitlist() {}  
  
    fn seat_at_table() {}  
  }  
  
  mod serving {  
    fn take_order() {}  
  
    fn serve_order() {}  
  
    fn take_payment() {}  
  }  
}
```

Restoran ile ilgili bir kütüphane ;

Front-of-house → ön hizmetler

Hosting → karşılama

Serving → hizmet



Root Module

`crate`

└─ `front_of_house` Parent Module

└─ `hosting` Child Module

└─ `add_to_waitlist`

└─ `seat_at_table`

└─ `serving` Child Module

└─ `take_order`

└─ `serve_order`

└─ `take_payment`



Modül Ağacındaki Bir Öğeye Başvurma Yolları

Absolute path → Crate roottan başlayan tam yoldur.

- Harici crate kullanılacaksa crate değimi ile başlar.

Relative path → Geçerli modülden başlar, self, süper ya da geçerli modüldeki bir tanımlayıcı kullanılır.

Her ik yol :: ile devam eder.

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



super:: ile Relative Path Oluşturma

super::Ulaşılabak_Eleman → Geçerli modül veya crate root yerine üst modülden başlayan relative yol oluşturur.

```
fn deliver_order() {}

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

    fn cook_order() {}
}
```



Hem struct hem de alanları ayrı ayrı ayarlanmalıdır.

```
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() {  
    let mut meal = back_of_house::Breakfast::summer("Rye");  
  
    meal.toast = String::from("Wheat");  
    println!("I'd like {} toast please", meal.toast);  
  
    // meal.seasonal_fruit = String::from("blueberries");  
}
```




Enum Yapısında Gizlilik - pub Kullanımı

Sadece enum tanımlamasında pub eklemek yeterlidir. Bütün öğelere ulaşılır.

```
mod back_of_house {  
    pub enum Appetizer {  
        Soup,  
        Salad,  
    }  
}  
  
pub fn eat_at_restaurant() {  
    let order1 = back_of_house::Appetizer::Soup;  
    let order2 = back_of_house::Appetizer::Salad;  
}
```

use ile Path Tanımlama



```
mod front_of_house {  
  pub mod hosting {  
    pub fn add_to_waitlist() {}  
  }  
}  
  
use crate::front_of_house::hosting;  
  
pub fn eat_at_restaurant() {  
  hosting::add_to_waitlist();  
}
```

```
use crate::front_of_house::hosting;  
  
mod customer {  
  pub fn eat_at_restaurant() {  
    hosting::add_to_waitlist();  
  }  
}
```

Mod dışında tanımlana use, mod için geçerli değildir.

Hata oluşur



Use içinde kullanılacak öge belirtilerek daha kolay erişim sağlanır.

```
use crate::front_of_house::hosting::add_to_waitlist;  
  
pub fn eat_at_restaurant() {  
    add_to_waitlist();  
}
```



" as " Değimi ile Farklı İsimler Verilebilir

```
use std::fmt::Result;
use std::io::Result as IoResult;

fn function1() -> Result {
    // --snip--
    Ok(())
}

fn function2() -> IoResult<()> {
    // --snip--
    Ok(())
}
```



pub use ile Importlar da Paylaşılabilir

```
pub use crate::front_of_house::hosting;
```




Harici Paket Kullanılımı

cargo.toml → paket isim ve sürümleri belirtilir.

rand = "0.8.5" → sonunda noktalama işareti yoktur.

KULLANIMI :

<https://crates.io/>

```
use rand::Rng;
```

```
....
```

```
let rastgele_sayı = rand::thread_rng().gen_range(1..=100);
```



Büyük Kullanım Listelerini Temizlemek için İç İçe Yolları Kullanma

```
use std::cmp::Ordering;  
use std::io;
```

Kullanımı aşağıdaki gibi tek bir satıra alınabilir.

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

```
use std::io;  
use std::io::Write;
```

Kullanımı aşağıdaki gibi tek bir satıra alınabilir.

```
use std::io::{self, Write};
```



Glob "*" Operatör

`use std::collections::*;` → collections içindeki bütün public öğeleri içe aktarır.



Kendi Paketini Oluşturma ve Kendi Projende Kullanmak

[dependencies] kısmına aşağıdaki ayarlar eklenir.

Yerel Dosyalarda ;
 crate_adı = { path = "../proje_klasörü_adı" }

Github'tan ;
 crate_adı = { git = "[https://.../paket_adı.git](\"https://.../paket_adı.git\")" }

Crate'in Kullanılacağı Projede;

```
extern crate crate_adı;
```

```
...
```

```
let degisken = crate_adı::altmoduladi::crate_ogesi();
```



<https://doc.rust-lang.org/book/ch07-05-separating-modules-into-different-files.html>



Celal AKSU

Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

<https://www.linkedin.com/in/cilaksu/>

<https://twitter.com/ksacil>

<https://www.youtube.com/@eemcs>