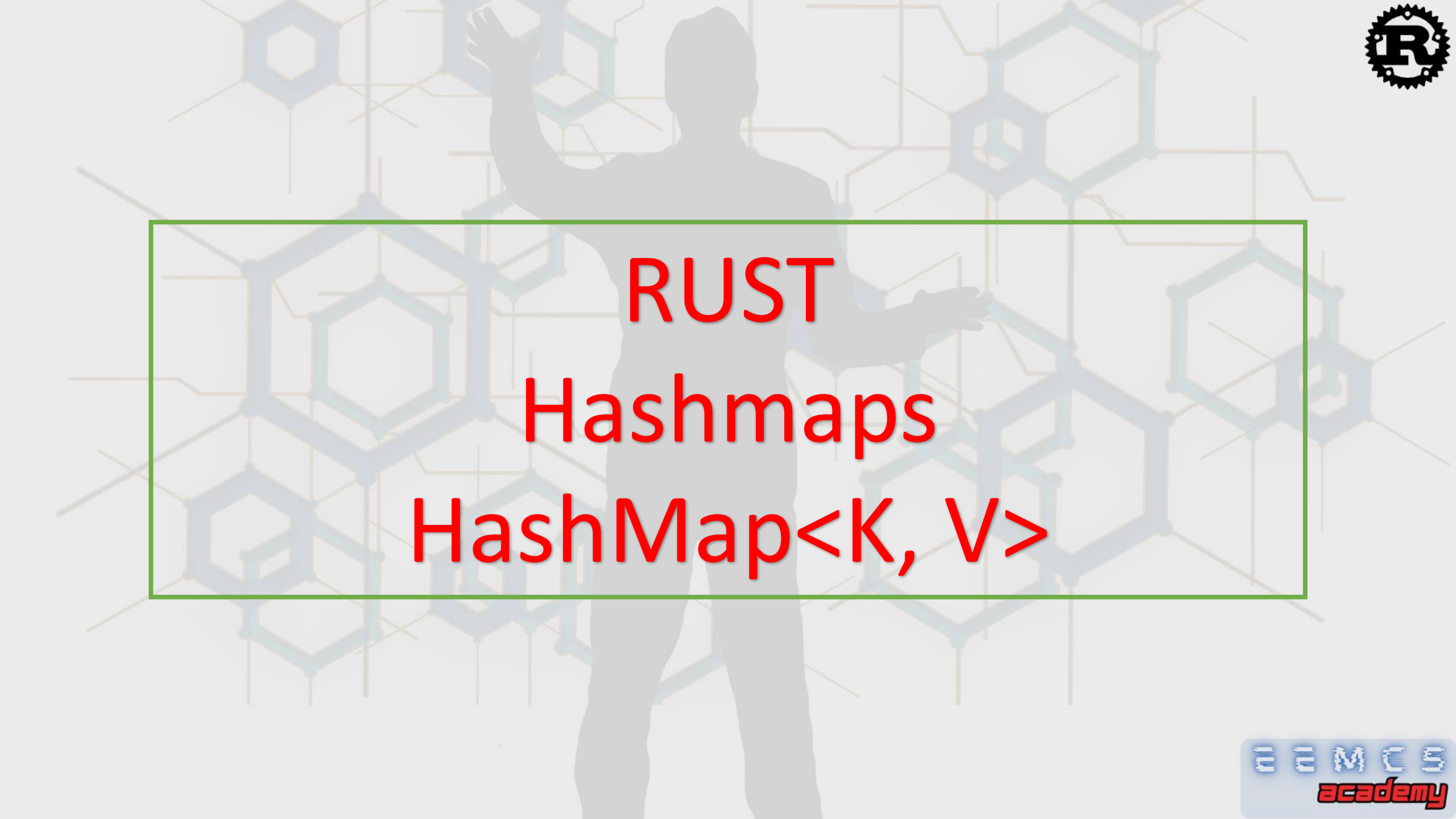

RUST
Hashmaps
HashMap<K, V>

HashMap Nedir?

HashMap<K, V>

- Anahtar (Key), Değer (Value) yapısındaki veri türüdür.
- Key ve Value birbirine bağlı bir yapıdır.
- Key ve Value verilerini hafızaya hasing (SipHash) fonksiyonu kayıt eder.
- K key olarak kullanılacak verinin tipini ifade eder. Ve bütün keyler bu tipte olmak zorundadır..
- Key verisi tek olmak zorundadır. Aynı keyden birden fazla olamaz
- V value olarak kullanılacak verinin tipini ifade eder. Bütün değerler bu veri tipinde olmak zorundadır.
- Veriler vector deki gibi indexlenmez.
- Veriye ulaşmak için key değeri kullanılır.

HashMap Oluşturmak :

Öncelikle **use std::collections::Hashmap;** ile içe aktarılmalıdır.

Boş HashMap Oluşturmak, new() ile

```
let mut hash_map_adı:HashMap<key_veri_tipi, value_veri_tipi> = HashMap::new();
```

```
let mut scores:HashMap<String, u32> = HashMap::new();
```

HashMap Verisini Yazdırma → {:?}

HashMap Oluşturmak :

Boş HashMap Oluşturmak, new() ile

```
let mut hash_map_adı = HashMap::new();
```

Bu yöntem kullanıldığında hashmap tipleri veri eklendikten sonra otomatik belirlenir.

HashMap'i Key-Value Değerleri Vererek Tanımlamak - (Dizileri kullanarak)

Let hashmap_adı = HashMap::from([(key, value), (key, value), ...]);

```
let mut hashmap2 = HashMap::from([("a", 10), ("b", 20)]);
```

HashMap'a Eleman Eklemek - insert() metodu

hasmap_adı.insert(KEY, VALUE)

→ Metod ile işlem yapıldıktan sonra, metod geriye veri döndürür. Bu veri işlenerek farklı algoritmalar çıkartılabilir.

```
scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Yellow"), 50);

println!("Hashmap verileri : {:?}", scores);
```

Hashmap verileri : {"Blue": 10, "Yellow": 50 } // output

HasMap Güncelleme veya Ekleme - insert()

HashMap'ta Key verisi tek olmak zorundadır.
insert() ile var olan key değeri ile yeni veri eklenince key karşılığındaki veri değiştirilmiş olur.

```
scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Blue"), 25);
println!("{:?}", scores);
```

İnsert() metodu ile veri eklenince, geriye Option<T> türünde değer gönderir.
Veri ilk kez eklenince None değerini geri gönderir.
Aynı key değerine, veri eklenmeye çalışılırsa eski değeri Some<T> türünde geri döndürür. T → Eski değerin tipini simgeler. Some(10) değeri gelir.

Sadece Key Varolmadığında Veri Ekleme - entry()

```
let mut scores = HashMap::new();
scores.insert(String::from("Blue"), 10);
scores.entry(String::from("Yellow")).or_insert(50);
scores.entry(String::from("Blue")).or_insert(50);
println!("{:?}", scores);
```

or_insert() metodu, eğer key verisi varsa,
key karşılığındaki değeri mutable referans tipinde (&mut V) döndürür.
Eğer key yoksa,
parametre olarak aldığı veriyi ekler ve yeni değerin mutable referansını döndürür.

Eski Değere Göre Veriyi Güncelleme

```
let text = "hello world wonderful world";
let mut map = HashMap::new();

for word in text.split_whitespace() {
     let count = map.entry(word).or_insert(0);
     *count += 1;
}

println!("{:?}", map);
```

```
{"world": 2, "hello": 1, "wonderful": 1}
```

split_whitespace() → Boşluğa göre metni kelimelere böler

entry() → key olmadığında, verilen değeri key olarak ekler

or_insert() → key ilk eklendiğinde value değerini 0-sıfır yapar ve bu değeri mutable referans olarak count değişkenine atar.

*count +=1 → value değeri mutable olduğu için
ve aynı değer referans olarak geri döndürüldüğü için
kelime key olarak eklendiğinde 0-sıfır olan value 1-bir olur
kelime key olarak ikinci kez kontrol edildiğinde de 1-bir olan değer 2 olur.

* → deference işlemi yapar yani verinin kendisine erişim sağlar

HasMap'lerde iterasyon işlemi keyfi olarak yapılmaktadır. Yani bir sıra yoktur. Çıktıda o yüzden sıralama yoktur.

HashMap Değerlerine Erişmek - get() metodu ile sayısal verilere erişim - hashmap_adı.get(key)

```
let team_name = String::from("Blue");
→team_name, key değerini belirlemek için tanımlanmıştır.

let score = scores.get(&team_name).copied().unwrap_or(0);

→ get(&K) ile veri alınır.
→ get() metodu Option<&V> tipinde veri döndürür.
→ Option<&V> tipini Option<V> tipine dönüştürmek için
copied() metodu kullanılır.
→ unwrap_or(0) ile, eğer dönen gerçek bir değer varsa bu
değer alınır; yoksa değer 0-sıfır olarak belirlenir.
```

Option<T> Nedir?

Özel tanımlı, enum yapısında bir veri türüdür.
İki adet değeri vardır.

Some(T)
None

Yukarıdaki gibi, veri döndüren yapılardan güvenli veri almak için kullanılır. Yani veri geliyorsa Some(T) değerini verir; eğer veri gelmiyorsa None değerini verir.

Böylece dönen veri yoksa bile program çökmez.

Ve iki durum match ile kontrol edilerek programda güvenli akış sağlanmış olur.

Some<T> → T : gelen verinin tipini temsil eder. Bu u16, String vb. herhangi bir veri tipi olabilir.

Some(5), Some("Rust") vb. şeklindedir.

```
let pogramlama_dili = Some("Rust");
```

Şeklinde de bir değişken tanımalaması yapılabilir. Bu verinin tipi Option<T> olur.

HashMap Değerlerine Erişmek - get() metodu ile String verilere erişim

get() metodunun kullanımı vectörler ile aynıdır.

```
    let mut hashmap2 = HashMap::from([
        ("ad", "Celal"),
        ("soyad", "Öğretmen"),
        ("meslek", "öğretmen"),
    ]);

    let veri1 = hashmap
        .get("ad")
        .clone()
        .unwrap_or(&String::from("Yok"))
        .to_string();
    println!("Gelen veri : veri1 {}", veri1);
```

match { } İle Gelen Verinin İşlenmesi :

get() metodunun kullanımı vectörler ile aynıdır.

```
let veri1 = hashmap.get("ad");
match veri1 {
    Some(veri) => println!("Gelen veri : {}", veri),
    None => println!("Gelen veri yok"),
}
```

```rust
let mut alınacak_veri = String::new();
let veri1 = hashmap.get("ad");
match veri1 {
    Some(veri) => {
        println!("Gelen veri : {}", veri);
        alınacak_veri = veri.clone();
    }
    None => println!("Gelen veri yok"),
}
println!("Alınan veri : {}", alınacak_veri);
```

For ile HashMap Değerlerine Erişim.

```
for (key, value) in &scores {
      println!("{key}: {value}");
}
```

Hash Mapler ve Ownership

Basit veri tipleri için (stack memory verileri) Copy trait çalışır.

Heap memory verileri için Move() olayı çalışır.
Heap memory verisi HashMap'a eklenirse sahibi (owner) HashMap olur. Diğer değişken devre dışı kalır.

```
let field_name = String::from("Favorite color");
let field_value = String::from("Blue");
let mut map:HashMap<String, String> = HashMap::new();
map.insert(field_name, field_value);

// insert işleminden sonra field_name, field_value değişkenleri geçersiz olur.
```

HashMap'ten Eleman Çifti Silme

```
scores.remove(&String::from("Blue"));

Silinecek olan key verisinin &-referans olarak
belirtildiğine dikkat ediniz.

Silinen veriyi Option<T> türünde geri döndürür.
Veri varsa Some<T>,
Veri yoksa None değerini döndürür.
```

Extra : Hashing Fonksiyonları

Varsayılan olarak SipHash adında bir hash fonksiyonu kullanır.
SipHash hash tablolarına karşı yapılan DoS saldırılarına karşı dirençlidir.
Siphash en hızlı hash algoritması değildir.

Farklı bir hasher algoritması bullanabilirsiniz.
Hasher, BuildHasher tarit'ini implement eden bir tiptir.

Struct Kullanarak HashMap Oluşturma:

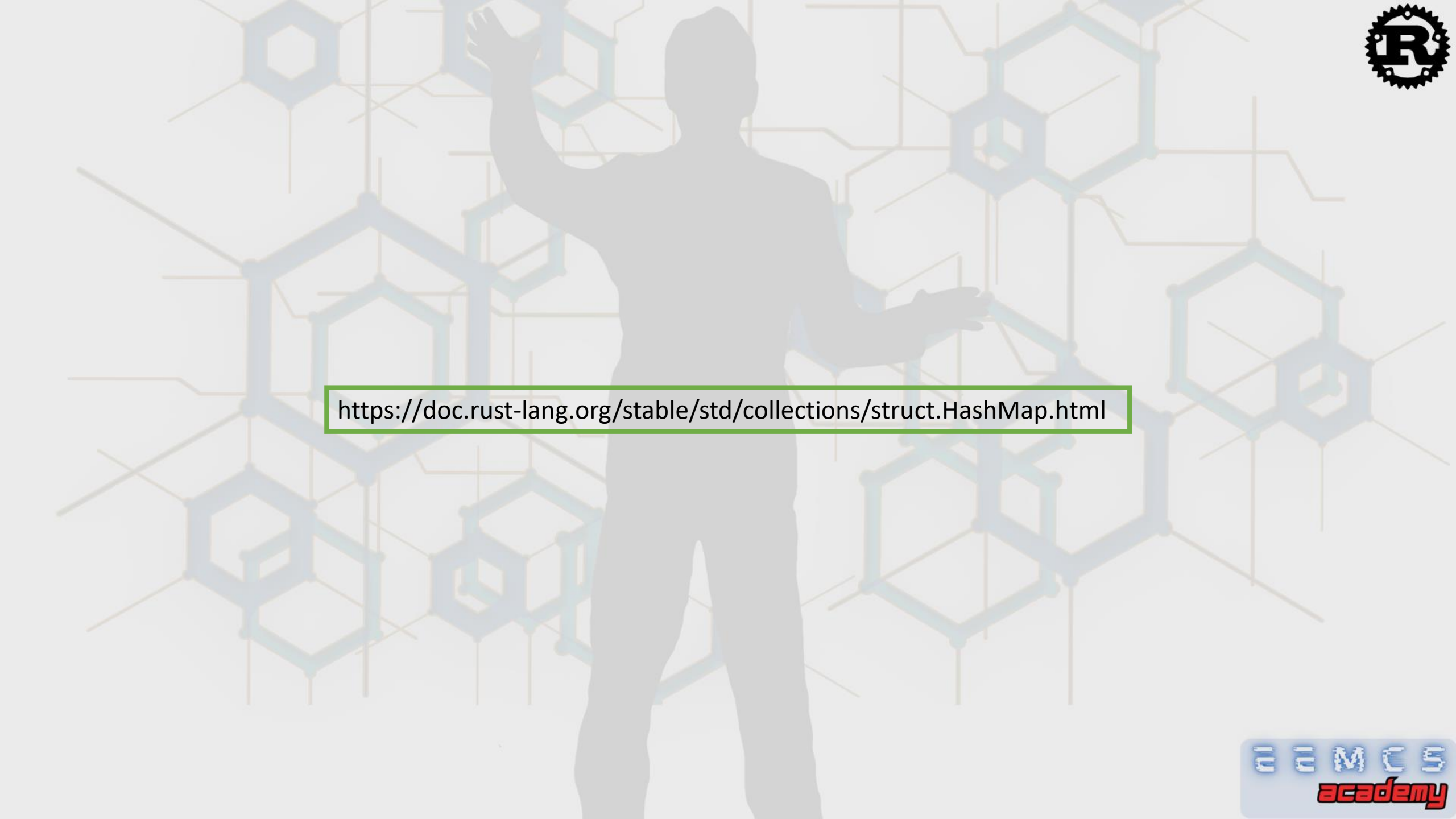
```
#![allow(unused)]
fn main() {
use std::collections::HashMap;

#[derive(Hash, Eq, PartialEq, Debug)]
struct Viking {
    name: String,
    country: String,
}

impl Viking {
    /// Creates a new Viking.
    fn new(name: &str, country: &str) -> Viking {
        Viking { name: name.to_string(), country: country.to_string() }
    }
}
```

```rust
// Use a HashMap to store the vikings' health points.
let vikings = HashMap::from([
    (Viking::new("Einar", "Norway"), 25),
    (Viking::new("Olaf", "Denmark"), 24),
    (Viking::new("Harald", "Iceland"), 12),
]);

// Use derived implementation to print the status of the vikings.
for (viking, health) in &vikings {
    println!("{viking:?} has {health} hp");
}
}
```



https://doc.rust-lang.org/stable/std/collections/struct.HashMap.html

Celal AKSU
Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

https://www.linkedin.com/in/cllaksu/

https://twitter.com/ksacll

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