



Universidad de Oviedo



# Arquitectura del Software

## Presentación Seminario 1 - Rust



ARQUITECTURA  
DEL SOFTWARE

2025-26

Jose E. Labra Gayo  
Pablo González  
Diego Martín Fernández  
Celia Melendi Lavandera

# ¿Qué es Rust?

## Resumen

- Es un lenguaje de programación compilado y memory safe
- Tiene funcionalidades de alto nivel
- Tiene rendimiento como un lenguaje de bajo nivel
- No tiene garbage collector
- Y muchas otras cosas más...

## ¿Y esto a mí que?



# ¿Cómo funciona Rust? (Desde la perspectiva de Java)

## 1. Creación de objetos

```
public class User {  
    private String name;  
    public User(String name) { this.name = name; }  
    public void greet() { System.out.println("Hi " + name); }  
}
```

```
pub struct User {  
    name: String, // Private by default  
}  
  
impl User {  
    pub fn new(name: String) -> Self { Self { name } }  
    pub fn greet(&self) { println!("Hi {}", self.name); }  
}
```

# ¿Cómo funciona Rust? (Desde la perspectiva de Java)

## 2. Interfaces

### Java

```
interface User {  
    String getRole();  
}
```

```
class Admin implements User {  
    public String getRole() {  
        return "Administrator";  
    }  
}
```

### Rust

```
trait User {  
    fn get_role(&self) -> String;  
}
```

```
struct Admin; // Just the data  
  
impl User for Admin { // Attach the "User" job to Admin  
    fn get_role(&self) -> String {  
        String::from("Administrator")  
    }  
}
```

# ¿Cómo funciona Rust? (Desde la perspectiva de Java)

## 3. Zero cost abstractions

### Java

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);

int sum = numbers.stream()
    .filter(n -> n % 2 == 0)
    .map(n -> n * 10)
    .reduce(0, Integer::sum);
```

### Rust

```
let numbers = vec![1, 2, 3, 4, 5, 6];

let sum: i32 = numbers.iter()           // 1. Create iterator
    .filter(|&n| n % 2 == 0)           // 2. Closure (lambda)
    .map(|n| n * 10)                   // 3. Transformation
    .sum();                           // 4. Consumer
```

# ¿Cómo funciona Rust? (Desde la perspectiva de Java)

- 4. No hay garbage collector

## Java

```
public class User {  
    String name;  
    public User(String name) { this.name = name; }  
}  
  
public void process() {  
    User u = new User("Alice");  
    // 'u' es una referencia. El objeto vive en el Heap.  
    System.out.println(u.name);  
}  
// Al terminar la función, la referencia 'u' muere,  
// ¡pero el objeto sigue en el Heap hasta que el GC pase!
```





# ¿Cómo funciona Rust? (Desde la perspectiva de Java)

- 4. No hay garbage collector

## Rust

```
struct User {  
    name: String,  
}  
  
fn process() {  
    let u = User { name: String::from("Alice") };  
    // 'u' es el DUEÑO de los datos.  
    println!("{}", u.name);  
}  
  
// <--- ¡AQUÍ! Al cerrar la llave, Rust llama automáticamente a 'drop'.  
// La memoria se libera en este microsegundo exacto. No hay GC.
```



Y mucho más



# ¿Cómo se estructura un proyecto en Rust?

```
my_project/
├── Cargo.toml      # The "Manifest" (Equivalent to pom.xml or build.gradle)
├── Cargo.lock      # Generated: exact versions of dependencies (like package-lock.json)
├── src/            # All source code lives here
│   ├── main.rs     # The entry point for an executable (contains main function)
│   ├── lib.rs       # The entry point for a library (if applicable)
│   └── models/      # A "Package" or "Module"
│       ├── mod.rs   # Defines the 'models' module
│       └── user.rs  # Code for the User struct
├── tests/          # Integration tests (External to your code)
│   └── integration.rs
├── examples/       # (Optional) Example code for library users
├── target/         # Compiled artifacts (Equivalent to /target/ in Maven)
└── .gitignore      # Automatically excludes 'target/'
```