Universidad de Oviedo

Software Architecture
Seminar 1 Presentation -Rust

2025-26

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

What is Rust?

Summary

- It's a compiled and memory safe programming languague
- Has high level functionalities
- Has a performance comparable to low level languages
- Does not have a garbage collector
- AND many other things...

And?

How does it work? Rust (From a Java perspective)

1. Object creation

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

```rust
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); }
}
```

How does it work? Rust(From a Java perspective)

2. Interfaces

Java

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

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

Rust

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

```rust
struct Admin; // Just the data

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

How does it work? Rust (From a Java perspective)

3. Zero cost abstractions

Java

```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

```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
```

How does it work? Rust (From a Java perspective)

- 4. There is no garbage collector

Java

```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!
```

How does it work? Rust (From a Java perspective)

- 4. There is no garbage collector

Rust

```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.
```

And much more

How is a project structured in 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/'
```