Lesson 8 Outline

- Recap Lesson 7
- Concurrency
- Threads
- Comunicazione threads
- Practice & Examples

https://github.com/RustRome/corso-rust

Error Handling

- Unrecoverable with panic!
- Recoverable with Result<T,E>
- Match for handling failure
- ? operator for propagation
- Custom errors with Enums + Conversion

Smart Pointers

- Box<T>
  - Allows immutable or mutable borrows checked at compile time
  - Single owner
- Rc<T>
  - Allows only immutable borrows checked at compile time
  - Multiple owners
- RefCell<T>
  - Allows mutable borrows checked at runtime.
  - Interior mutability pattern

Testing

- Integrated in Rust
- #[test]
- assert!
- assert_eq!
- cargo test

Concurrency is hard

Fearless Concurrency

Threads

Threads

- Multiple pieces of code at the same time
- Race Conditions
- Deadlocks
- Hard to reproduce and fix reliably

Threads spawn

```rust
use std::thread;


fn main() {
    thread::spawn(|| {
        println!("Hello from new Thread")
    });

    println!("Hello from the main Thread");

}
```

Threads sleep

```rust
use std::thread;
use std::time::Duration;

fn main() {
    thread::spawn(|| {
        println!("Hello from new Thread")
    });


    thread::sleep(Duration::from_millis(200));
    println!("Hello from the main Thread");

}
```

Threads join

```rust
use std::thread;

fn main() {
    let handle = thread::spawn(|| {
        println!("Hello from new Thread")
    });

    println!("Hello from the main Thread");

    handle.join();
}
```

Threads join with return

```rust
use std::thread;

fn main() {
    let handle = thread::spawn(|| {
        10
    });

    println!("Result from thread: {}", handle.join().unwrap());

}
```

Threads: Vec sum example

```
use std::thread::{self,JoinHandle};

fn main() {

    let vec = vec![1,10,32];

    let handle : JoinHandle<i32> = thread::spawn(|| {
        vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

}
```

Threads: Vec sum example error

```
 Compiling playground v0.0.1 (/playground)
error[E0373]: closure may outlive the current function, but it borrows `vec`, which is owned
by the current function
 --> src/main.rs:7:50
  |
7 |     let handle : JoinHandle<i32> = thread::spawn(|| {
  |                                                  ^^ may outlive borrowed value `vec`
8 |         vec.iter().sum()
  |         --- `vec` is borrowed here
  |
note: function requires argument type to outlive `'static`
 --> src/main.rs:7:36
  |
7 |       let handle : JoinHandle<i32> = thread::spawn(|| {
  |  ____________________________________^
8 | |         vec.iter().sum()
9 | |     });
  | |______^
help: to force the closure to take ownership of `vec` (and any other referenced variables),
use the `move` keyword
  |
7 |     let handle : JoinHandle<i32> = thread::spawn(move || {
  |                                                  ^^^^^^^
```

Threads: Vec sum example fix

```
use std::thread::{self,JoinHandle};

fn main() {

    let vec = vec![1,10,32];

    let handle : JoinHandle<i32> = thread::spawn(move || {
        vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

}
```

Threads: Vec sum v2

```rust
use std::thread::{self,JoinHandle};

fn main() {

    let vec = vec![1,10,32];

    let handle : JoinHandle<i32> = thread::spawn(move || {
        vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

    println!("Vec size {}", vec.len());

}
```

Threads: Vec sum v2 - error

```
 Compiling playground v0.0.1 (/playground)
error[E0382]: borrow of moved value: `vec`
  --> src/main.rs:13:29
   |
5  |     let vec = vec![1,10,32];
   |         --- move occurs because `vec` has type `std::vec::Vec<i32>`, which does not
implement the `Copy` trait
6  |
7  |     let handle : JoinHandle<i32> = thread::spawn(move || {
   |                                                  ------- value moved into closure here
8  |         vec.iter().sum()
   |         --- variable moved due to use in closure
...
13 |     println!("Vec size {}", vec.len());
   |                             ^^^ value borrowed here after move
```

Threads: Vec sum v2 - fix

```rust
use std::thread::{self,JoinHandle};
use std::sync::Arc;

fn main() {

    let vec = Arc::new(vec![1,10,32]);

    let new_vec = vec.clone();

    let handle : JoinHandle<i32> = thread::spawn(move || {
        new_vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

    println!("Vec size {}", vec.len());

}
```

Do not communicate by sharing memory;
instead, share memory by communicating.

Channels

Channels

```
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();
}
```

Channels send

```
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
    });
}
```

Channels receive

```
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
    });

    let received = rx.recv().unwrap();
    println!("Got: {}", received);
}
```

Channels Ownership Transference

```rust
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
        println!("val is {}", val);
    });

    let received = rx.recv().unwrap();
    println!("Got: {}", received);
}
```

Channels Ownership Transference

```
rror[E0382]: use of moved value: `val`
  --> src/main.rs:10:31
   |
9  |         tx.send(val).unwrap();
   |                 --- value moved here
10 |         println!("val is {}", val);
   |                               ^^^ value used here after move
   |
   = note: move occurs because `val` has type `std::string::String`, which does
not implement the `Copy` trait
```

Channels multiple senders

```
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    let first_tx = tx.clone();
    thread::spawn(move || {
        let val = String::from("hi from thread 1");
        first_tx.send(val).unwrap();

    });

    thread::spawn(move || {
        let val = String::from("hi from thread 2");
        tx.send(val).unwrap();

    });

    for received in rx {
        println!("Got: {}", received);
    }

}
```

Shared Memory

Threads: Vec sum v3

```rust
use std::thread::{self,JoinHandle};
use std::sync::Arc;

fn main() {

    let vec = Arc::new(vec![1,10,32]);

    let new_vec = vec.clone();

    let handle : JoinHandle<i32> = thread::spawn(move || {
        new_vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

    println!("Vec size {}", vec.len());

}
```

Threads: Vec sum v3

```rust
use std::thread::{self,JoinHandle};
use std::sync::Arc;

fn main() {

    let vec = Arc::new(vec![1,10,32]);

    let new_vec = vec.clone();

    let handle : JoinHandle<i32> = thread::spawn(move || {
        new_vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

    println!("Vec size {}", vec.len());

}
```

Threads: Vec sum v3

```rust
use std::thread::{self,JoinHandle};
use std::sync::Arc;

fn main() {

    let vec = Arc::new(vec![1,10,32]);

    let new_vec = vec.clone();

    let handle : JoinHandle<i32> = thread::spawn(move || {
        new_vec.push(10);
        new_vec.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

    println!("Vec size {}", vec.len());

}
```

Threads: Vec sum v3 - error

```
Compiling playground v0.0.1 (/playground)
error[E0596]: cannot borrow data in an `Arc` as mutable
  --> src/main.rs:11:9
   |
11 |         new_vec.push(10);
   |         ^^^^^^^ cannot borrow as mutable
   |
   =
```

Threads: Vec sum v3 - fix

```rust
use std::thread::{self,JoinHandle};
use std::sync::{Arc,Mutex};


fn main() {

    let vec = Arc::new(Mutex::new(vec![1,10,32]));

    let new_vec = vec.clone();

    let handle : JoinHandle<i32> = thread::spawn(move || {
        let mut data = new_vec.lock().unwrap();
        data.push(10);
        data.iter().sum()
    });

    println!("Result from thread: {}", handle.join().unwrap());

    println!("Vec size {}", vec.lock().unwrap().len());

}
```

std::sync::*

- Arc<T>
- Mutex<T>
- RwLock<T>
- Channels
- ....

Thank You

Rust Language cheat sheet

Rust Roma Meetup

https://www.meetup.com/it-IT/Rust-Roma/