Rust 2023

clase 5

Temario

- Generics
- Traits
- POO

Generics

Generics: ¿Para qué sirven?

El tipo generic se utiliza para generalizar implementaciones, permite mayor flexibilidad en el código

Generics: ejemplo I

```
#[derive(Debug)]
struct Punto{
    x:i32,
    y:i32,
}
fn main(){
    let p1 = Punto{x:1,y:2};
    println!("{:?}", p1);
}
```

Generics: ejemplo I

```
#[derive(Debug)]
struct Punto{
    x:i32,
    y:i32,
}
fn main(){
    let p1 = Punto{x:1,y:2};
    let p2 = Punto{x:10.5,y:2};
    println!("{:?} {:?}", p1, p2);
}
```

```
error[E0308]: mismatched types
  --> src/main.rs:27:22
   |
27 |     let p2 = Punto{x:10.5,y:2};
   |                      ^^^^ expected `i32`, found floating-point number
```

Generics: ejemplo I

```
struct Punto<T>{
    x:T,
    y:T,
}
fn main(){
    let p1 = Punto{x:1,y:2};
    let p2 = Punto{x:10.5,y:2.0};
}
```

Generics: ejemplo I

```
struct Punto<T>{
    x:T,
    y:T,
}
fn main(){
    let p1 = Punto{x:1,y:2};
    let p2 = Punto{x:10.5,y:2};
}
```

```
error[E0308]: mismatched types
  --> src/main.rs:10:29
   |
10 |     let p2 = Punto{x:10.5,y:2};
   |                             ^ expected floating-point number, found integer
```

Generics: ejemplo I

```
struct Punto<T, V>{
    x:T,
    y:V,
}
fn main(){
    let p1 = Punto{x:1,y:2};
    let p2 = Punto{x:10.5,y:2};
}
```

Generics: ejemplo I

```
#[derive(Debug)]
struct Punto<T>{
  x:T,
  y:T,
}
fn main(){
  let p1 = Punto{x:1,y:2};
  let p2 = Punto{x:10,y:2};
  println!("{:?}", p1);
  println!("{:?}", p2);
}
```

Generics: ejemplo I warning!!!

```
fn main(){
    let p1 = Punto{x:1,y:2};
    let p2 = Punto{x:10,y:2};
    let p_esp = Punto{x:p1, y:p2};
    println!("{:?}", p_esp);
}
```

Generics: ejemplo de caja

```
struct Caja {
    dato:i32,
    estado:bool,
}
impl Caja {
    fn new(dato:i32) -> Caja{
        Caja { dato, estado:false}
    }
    fn abrir(&mut self)->i32{
        self.estado = true;
        self.dato
    }
    fn cerrar(&mut self){
        self.estado = false;
    }
}
```

Generics: otro ejemplo

```
fn main(){
    let mut caja = Caja::new(9);
    caja.abrir();
    caja.cerrar();
}
```

Generics: otro ejemplo

```
fn main(){
    let mut listado_de_compras = Vec::new();
    listado_de_compras.push((1, "Jabon de manos"));
    listado_de_compras.push((2, "Detergente"));
    let mut caja = Caja::new(listado_de_compras);
    caja.abrir();
    caja.cerrar();
}
```

```
error[E0308]: mismatched types
  --> src/main.rs:24:30
   |
24 |     let mut caja = Caja::new(listado_de_compras);
   |                    ---------- ^^^^^^^^^^^^^^^^^^^ expected `i32`, found struct `Vec`
   |                    |
   |                    arguments to this function are incorrect
   |
   = note: expected type `i32`
              found struct `Vec<({integer}, &str)>`
```

Generics: otro ejemplo

```
struct Caja <T>{
    dato:T,
    estado:bool,
}
impl<T> Caja<T> {
    fn new(dato:T) -> Caja<T>{
        Caja { dato, estado:false}
    }
    fn abrir(&mut self)->&T{
        self.estado = true;
        &self.dato
    }
    fn cerrar(&mut self){
        self.estado = false;
    }
}
```

Generics

veamos un caso más complejo...