Comparison of Rust and Java

Tomáš Janecký Michael Vlach

BARX FX

18th October 2022

About Rust

- introduced in 2010
- compiler guaranties memory, thread and type safety by default (can be violated with `unsafe{}` block)
- Haskell inspired polymorphism
- memory is freed automatically without garbage collector
- no concept of Null pointer, `Option` type instead (similar to `Optional<T>`)

Rust users

- Will be added to Linux kernel 6.1
- Microsoft Azure (C/C++ deprecated)
- JP Morgan
- used in Mozilla Firefox

Memory safety

```java
class Main {
 public static Integer
                  f(Integer x){
        return null;
 }
 public static void
          main(String[] args){
  Integer x = 10;
  x = f(x);
  System.out.
    println(x.intValue());
 }
}
```

```rust
fn f(x: &mut i32)->Option<i32>{
  None
}

fn main() {
  let mut x = 10;
  x = *f(&mut x);//has to be
                 //matched
  println!("{}", x);
}
```

```
error[E0614]: type `Option<i32>` cannot be dereferenced
--> src/main.rs:7:9

        x = *f(&mut x);
```

Memory safety 2

```
class Main {
 public static void
       main(String[] args){
  int[] numbers =
       {10, 20, 30};
  System.out.
       println(numbers[3]);
    }
}
```

```
fn main() {
  let numbers = [10, 20, 30];
  println!("{}", numbers[3]);
}
```

```
 println!("", numbers[3]);
              ^^^^^^^^^^ index out of bounds: the length is 3 but
 the index is 3
```

Thread safety

```
class Main {
 public static void f(
                    int[] x){
  for(int i=0 ; i<100 ; i++){
   x[0] += 1;
  }
 }
 public static void main(
              String[] args) {
  int[] x = new int[1];
  Thread t1 =
      new Thread(() -> f(x));
  Thread t2 =
      new Thread(() -> f(x));
  t1.start();
  t2.start();
  try {//join and println...
```

```
use std::{thread, cell::RefCell,
                          rc::Rc};
fn f(x: Rc<RefCell<i32>>){
  for _i in 1..100 {
    *x.borrow_mut() += 1;
  }
}
fn main() {
 let x =
     Rc::new(RefCell::new(0));
 let t1 =
     thread::spawn(||{ f(x)});
 let t2 =
     thread::spawn(||{ f(x)});
 t1.join().unwrap();
 t2.join().unwrap();
 println!("{:?}", x);
}
```

Thread safety cont.

```
let t1 = thread::spawn(|| { f(x)});
         ^^^^^^^^^^^^^^ -- within this
         |
        `Rc<RefCell<i32>>` cannot be sent between threads safely
note: required because it's used within this closure


     let t1 = thread::spawn(|| { f(x)});
                            ^^
note: required by a bound in `spawn`
```

More Information

- The Rust Programming Language - book
- More about safety of Rust
- Mozilla Rust foundation

Thank you!