Programowanie obiektowe i równoległe

5 grudnia 2025

Instrukcja

Zadanie zaimplementuj jako bibliotekę. Kod startowy dołącz do projektu jako lib.rs. Dopuszczalne jest użycie tylko biblioteki standardowej. Szczegółowy opis działania funkcji znajdziesz w komentarzach w kodzie startowym.

1. Zaimplementuj wszystkie elementy z sekcji *Funkcjonalność*.
2. Napraw wszystkie ostrzeżenia kompilatora.
3. Sprawdź, czy program przechodzi testy (`cargo test`).
4. Użyj `cargo clippy`, aby znaleźć typowe problemy i je poprawić.

Funkcjonalność

1. (Polimorfizm przez cechy) Zaimplementuj funkcje sumujące pola figur:
   - `total_area_generic<T: Shape>(items: &[T]) -> f64`—zwraca sumę `area()` wszystkich elementów przez statyczne dyspozycje.
   - `total_area_dyn(items: &[Box<dyn Shape>]) -> f64`—popraw sygnaturę i zwróć sumę pól przez dynamiczną dyspozycję.
2. (Obiektowość cech) `Transform` nie jest *object safe*. Wyjaśnij dlaczego i popraw definicję tak, by była (np. rezygnując z ogólności metody `apply`). Następnie:
   - Zaimplementuj `Transform` dla `Add` (dodaje `k`, `name() == "add"`) i `Mul` (mnoży przez `k`, `name() == "mul"`).
   - Zaimplementuj `apply_all_dyn(seq: &mut [f64], t: &dyn Transform)`—zastosuj transformację do wszystkich elementów w miejscu.
3. (Downcasting z `Any`) Zaimplementuj `sum_all_i32(boxes: &[Box<dyn Any>]) -> i32`—zwróć sumę tylko tych wartości, które są typu `i32` (inne zignoruj).
4. (Wątki—spawn/join) `spawn_sum(v: Vec<i32>)`—uruchom wątek liczący sumę elementów, a następnie wypisz wynik w wątku wywołującym po `join` (nie w wątku potomnym).
5. (Wątki—`thread::scope`) `sum_scoped(parts: &[&[i32]]) -> i32`—uruchom osobny wątek dla każdej części, każdy zwraca sumę częściową; wątek główny sumuje wyniki i zwraca całość.
6. (Zatrucie `Mutex`) `parallel_increment(n_threads, iters) -> i64`—licznik w `Arc<Mutex<i64>>`, każdy wątek wykonuje `iters` inkrementacji; wątki o indeksach parzystych panikują w ostatniej iteracji. Obsłuż zatrucie muteksu przy odblokowaniu, wykonaj `join` na wszystkich wątkach, wypisz indeksy wątków, które spanikowały, zwróć końcową wartość licznika.
7. (Kanały `mpsc`) `pipeline(n, threads) -> i32`—utwórz kanał i `threads` producentów wysyłających kolejno `1..=n`; *oddzielny* wątek agregujący (nie główny) sumuje wartości i wynik funkcji to policzona suma.

Wymagania

- Kod kompiluje się ze stabilnym kompilatorem Rust.
- Brak ostrzeżeń kompilatora i `clippy` (w tym o nieużywanych elementach).
- Wszystkie testy przechodzą.

Ocena (3 pkt)

- 1 pkt: Praca w trakcie laboratorium.
- 1 pkt: Pełna funkcjonalność (implementacja zgodna z wymaganiami).
- 1 pkt: Prezentacja rozwiązania i odpowiedzi na pytania prowadzącego.