

```
1 use std::collections::VecDeque;
2 use std::sync::{Arc, Mutex};
3
4 // Task Trait
5 trait Task {
6     fn execute(&self) -> usize;
7 }
8
9 // SumTask struct
10 struct SumTask {
11     n1: usize,
12     n2: usize,
13 }
14
15 impl SumTask {
16     fn new(n1: usize, n2: usize) -> Self {
17         Self { n1, n2 }
18     }
19 }
20
21 impl Task for SumTask {
22     fn execute(&self) -> usize {
23         self.n1 + self.n2
24     }
25 }
26
27 // LenTask struct
28 struct LenTask {
29     s: String,
30 }
31
32 impl LenTask {
33     fn new(s: String) -> Self {
34         Self { s }
35     }
36 }
37
38
```

```

39 impl Task for LenTask {
40     fn execute(&self) -> usize {
41         self.s.len()
42     }
43 }
44
45 // Shared task queue type
46 type TaskQueue = Arc<Mutex<VecDeque<Box<dyn Task>>>>>;
47
48 // Tasker struct
49 struct Tasker {
50     queue: TaskQueue,
51 }
52
53 impl Tasker {
54     // Create a new Tasker
55     fn new() -> Self {
56         Self {
57             queue: Arc::new(Mutex::new(VecDeque::new())),
58         }
59     }
60
61     // Schedule a task
62     fn schedule_task(&mut self, task: Box<dyn Task>) {
63         let mut queue = self.queue.lock().unwrap();
64         queue.push_back(task);
65     }
66
67     // Create a new Tasker linked to the current queue
68     fn get_tasker(&self) -> Tasker {
69         Tasker {
70             queue: Arc::clone(&self.queue),
71         }
72     }
73
74     // Create an Executer linked to the current queue
75     fn get_executer(&self) -> Executer {
76         Executer {
77             queue: Arc::clone(&self.queue),
78         }
79     }

```

```
80 | }
81 |
82 | // Executer struct
83 | struct Executer {
84 |     queue: TaskQueue,
85 | }
86 |
87 | impl Executer {
88 |     // Execute the next task in the queue
89 |     fn execute_task(&mut self) -> Option<usize> {
90 |         let mut queue = self.queue.lock().unwrap();
91 |         queue.pop_front().map(|task| task.execute())
92 |     }
93 | }
```