Code Review & Fixes

Overview

A code review and correction of a Rust-based user/course management system. Focuses on enums, structs, and method implementations.

🚧 Original Intent

What You Were Trying to Do

1. Define Roles
Created `Role` enum with variants:

```
enum Role { Client, Admin, Guest }
```

2. Define Courses
Created `Course` enum with variants holding data:

```
enum Course {
    Rust(Option<Vec<String>>),
    Html(Vec<String>),
    Java(Vec<String>),
}
```

3. Define Users
Created `User` struct:

```
struct User {
    name: String,
    role: Role,
    class: Course,
}
```

4. Define Modules
Created `Modules` enum for course-specific modules:

```
enum Modules {
    Rust(String),
    Html(String),
    Java(String),
}
```

5. Implement User Methods

- `add_course`: Add course + modules to user
- `create`: User constructor
- `get_name`, `get_role`, `get_user`: Info retrieval methods

🔥 Where Things Went Wrong

Syntax Errors

- **Course Enum**
  ✗ Invalid: `Option<Vec<String>, None>`
  ✅ Fixed: `Option<Vec<String>>`
- **Match Arms**
  ✗ Incomplete: `Modules::Rust("Rust for beginners") => ,`
  ✅ Fixed: Proper pattern matching
- **Field Mismatch**
  ✗ Referenced `self.course` but field is `class`

Logical Errors

- **Return Type Mismatch**
  ✗ `add_course` returned `Vec<String>` instead of `Course<String>`
- **Type Name Mismatch**
  ✗ `Courses` vs `Course` in struct definition

Incomplete Implementation

- **Unfinished `add_course`**
- **Empty `main` function**

🛠 Corrected Code

```
enum Role {
    Client,
    Admin,
    Guest,
}

enum Course {
    Rust(Option<Vec<String>>),
    Html(Vec<String>),
    Java(Vec<String>),
}
```

```rust
struct User {
    name: String,
    role: Role,
    class: Course,
}

enum Modules {
    Rust(String),
    Html(String),
    Java(String),
}

impl User {
    fn add_course(&mut self, course: Course, module: Modules) {
        match &mut self.class {
            Course::Rust(Some(modules)) => {
                if let Modules::Rust(module_name) = module {
                    modules.push(module_name);
                }
            }
            Course::Html(modules) => {
                if let Modules::Html(module_name) = module {
                    modules.push(module_name);
                }
            }
            Course::Java(modules) => {
                if let Modules::Java(module_name) = module {
                    modules.push(module_name);
                }
            }
            _ => {
                println!("Invalid course or module.");
            }
        }
    }

    fn create(name: String, role: Role, class: Course) -> Self {
        Self { name, role, class }
    }

    fn get_name(&self) -> &String {
        &self.name
    }

    fn get_role(&self) -> String {
        match self.role {
            Role::Client => String::from("Client"),
            Role::Admin => String::from("Admin"),
            Role::Guest => String::from("Guest"),
        }
    }

    fn get_user(&self) {
```

```
        println!("Name: {}", self.get_name());
        println!("{}, is a {} user", self.get_name(), self.get_role());
    }
}

fn main() {
    let mut user = User::create(
        String::from("Alice"),
        Role::Client,
        Course::Rust(Some(vec![String::from("Introduction to Rust")])),
    );

    user.get_user();

    user.add_course(
        Course::Rust(Some(vec![])),
        Modules::Rust(String::from("Rust for Beginners")),
    );

    if let Course::Rust(Some(modules)) = &user.class {
        println!("Rust Modules: {:?}", modules);
    }
}
```

📝 Explanation of Changes

1. Fixed Course Enum

```
- Fruits(Option<Vec<String>, None>)
+ Fruits(Option<Vec<String>>)
```

2. Field Name Correction

```
- struct User { class: Courses }
+ struct User { class: Course }
```

3. add_course Implementation

- Added proper module handling with type-safe matching:

```
match &mut self.class {
    Course::Rust(Some(modules)) => {
        if let Modules::Rust(module_name) = module {
            modules.push(module_name);
        }
```

```
    }
    // ... other variants
}
```

4. Demo main() Function

- Added concrete usage example:

```
let mut user = User::create(/* ... */);
user.add_course(/* ... */);
```

📚 Documentation

Enums Overview

Enum	Variants	Data Type
`Role`	Client, Admin, Guest	None
`Course`	Rust, Html, Java	`Option<Vec<String>>`
`Modules`	Rust, Html, Java	`String`

User Struct Methods

Method	Description
`add_course`	Adds module to matching course type
`create`	Constructor for new users
`get_user`	Prints formatted user info

🔜 Next Steps

1. Add error handling for module/course mismatches
2. Implement multiple courses per user
3. Add unit tests
4. Extend with course removal functionality