Student Name: _________________________________ Student ID: __________________ TA: _______________________

Rust Lab 13 – Data Encapsulation & Pattern Matching

1/10/2025

Lab 1: BankAccount (Encapsulation & Methods)

Goal: Practice module *privacy*, *associated functions, and self / &self / &mut self*.

Task: Create a bank module **BankAccount** with private fields owner: String, balance: u64. And method:

- `new(owner: String) -> BankAccount (balance starts at 0)`
- `deposit(&mut self, amount: u64) (amount 0 allowed; just no change)`
- `withdraw(&mut self, amount: u64) -> Result<(), String>`
- `balance(&self) -> u64`
- `owner(&self) -> &str`

Constraints:

- Disallow negative money (use u64).
- withdraw must fail if amount > balance.

Demonstrate (print to console):

1. Create account for Alice; make two deposits totaling ≥200.
2. **Call styles you must show (exactly these):**
   - deposit **once** using **dot notation**.
   - deposit **once** using **fully qualified syntax** (i.e., BankAccount::deposit).
   - withdraw **once** using **dot notation** (try over-withdraw that returns the error; print the error string).
3. Print final owner and balance.

TA Check: ____

Lab 2: Command Handler (Pattern Matching)

Goal: Developing a input system that interpret keyboard events accurately using idiomatic Rust pattern matching**.**

Tasks: - Create an array of characters: ['q', 'a', '7', 'x', '%', ‘9’, ‘A’, ‘d’].

- Write a function that takes a character and **uses a single match statement** to return a command string as follows:
  - Return "quit" if the character matches the constant **QUIT** which is 'q'.
  - Return "move" for any of 'a', 's', 'w', or 'd'.
  - Return "digit" for any character between '0' and '9'.
  - Return "lowercase" for any other lowercase letter using a guard.
  - Return "_other" for all other cases.
- For each character in the array, call your function and print the returned command string, one per line in the original array order.

TA Check: ____

Lab 3: Destructuring & @ Bindings

Goal: Enhance a graphics and parsing subsystem to classify spatial points and different input tokens, using rich pattern-matching techniques.

Task: - Data Structure: Create a vector containing exactly eight coordinate pairs, where each pair is a tuple of two integers ((i32, i32)). The order and values should be fixed and written directly in your code.

- Function: Write a function that receives a coordinate pair (tuple of two i32), and uses a single match expression—with tuple destructuring, guards, and at bindings— to return one of the following strings:
  - ○"I" if the point is in the first quadrant (x > 0 && y > 0)
  - ○"II" if the point is in the second quadrant (x < 0 && y > 0)
  - ○"III" if the point is in the third quadrant (x < 0 && y < 0)
  - ○"IV" if the point is in the fourth quadrant (x > 0 && y < 0)
  - ○"axis" for all other cases (if either x == 0 or y == 0)

Process: For each of the eight coordinate pairs in your vector, call this function and print the result—one output line per coordinate—strictly in the same order as the vector.

TA Check: ____

Lab 4: let else, while let, if let

Task: Implement the following:

```
1. pub fn first_hex_digit(maybe: Option<String>) -> Result<u32, String>
2. pub fn pop_all(s: &mut String) -> Vec<char>
3. pub fn print_parse_u8(s: &str)
```

Behavior Requirements

1. Function: first_hex_digit
   - If maybe is None, use let ... else to early return Err("none").
   - If the string is empty, return Err("empty").
   - If the first character is not a valid hexadecimal digit ([0-9a-fA-F]), return Err("not-hex").
   - On success, return Ok(value) where value is the numeric value of the first hex digit:
     - 9 maps to 0-9.
     - a-f or A-F maps to 10-15 (e.g., A or a → 10, F or f → 15).
2. Function: pop_all
   - Use while let with String::pop() to drain characters from the string.
   - Collect and return the popped characters in the order they were removed (last-to-first).
   - The input string must be empty after the operation.
3. Procedure: print_parse_u8
   - Define a helper function to parse s into an Option<u8>.
   - Use if let to print "n=<value>" only if parsing succeeds.
   - Do not print anything if parsing fails.

Demo Requirements

1. For first_hex_digit:
   - Call with Some("BEEF"), Some(""), and None.
   - Print the returned Result values.
2. For pop_all:
   - Start with the string "abc123".
   - Call pop_all, print the returned vector, and verify the original string is empty.
3. For print_parse_u8:
   - Call with "42" and "x".

Unit Tests

- Test first_hex_digit for:
  - Valid hex input (e.g., returns Ok(11) for "BEEF").
  - Empty string input (returns Err("empty")).
  - Non-hex first character (returns Err("not-hex")).
  - None input (returns Err("none")).
- Test pop_all to ensure:
  - The input string is empty after execution.
  - The returned vector contains the correct characters in pop order.

Constraints

- Do not use unwrap or expect.
- Do not use manual loop { ... break } for pop_all (use while let instead).